81

81 / 81

81 / 81

VIC-20 commodore COMPUTER

POCKET PET

POCKET PET NOTIZIE

POCKET PET
anno 1 - numero 4-5
numero unico in attesa
di autorizzazione

Redazione:
Harden S.p.A.
Via Pirelli 11
Milano

Direttore responsabile
Gloriano Rossi

Sommario

Questo numero di
POCKET PET
e' stato impaginato e
composto in redazione
con l'ausilio di
PET-CBM 4032
PET-CBM 8032
PET-CBM 3022
PET-CBM 4022
PET-CBM 8024
PET-CBM 8027
e con :
Wordpro 3.2
Wordcarft 80

Hanno collaborato a questo numero :

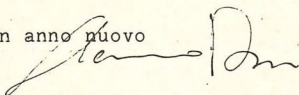
Alessandro de Simone
Gloriano Rossi
Massimo Rossi
Paolo Caletti
Riccardo Scotti
Roberto Sozzani
Stefano Miari

EDITORIALE

Numero 0, 1, 2·3 e 4·5

E passato un anno dall'uscita dell'edizione zero di POCKET PET.
A quel tempo erano stati promessi sei edizioni della rivista, e tante ne hanno visto la luce nell'arco del 1981.

Quali le promesse? Quali le propettive per il 1982?

POCKET PET, come il computer a cui e' dedicato, si espande e migliora sempre di piu' con il passare del tempo, pur mantenendo le medesime caratteristiche.

Il 1982 vedra' delle pagine sempre piu' interessanti per i possessori di un Commodore; articoli dedicati all'informazione, ai PET-CBM, al VIC20 e a tutti i prodotti Harden-Commodore.

Come e' suo uso e costume, la Harden S.p.A. proprio per meglio servire tutti i possessori presenti, passati e futuri di un Commodore, a partire dal prossimo anno altre sorprese inerenti al suo organo ufficiale: POCKET PET.

E per concludere:

Fra le pagine di questo numero doppio di fine d'anno, proprio per augurare un buon Natale, la redazione e la Harden S.p.A. offre a tutti i lettori un alberello un po' originale. E.....

buon anno nuovo

SSEMBLER PER TUTTI

di Alessandro de Simone

Da questa puntata anche se come detto precedentemente continuero' a riportare i programmi in LM da inserire mediante il MONITOR TIM trascrivero gli stessi scritti pero' in ASSEMBLER anche per .... onorare il nome della presente rubrica.

Come il lettore avra' il modo di accorgersi, i programmi Assembler sono molto piu' chiari e comprensibili e consentono, perfino a chi non possiede il programma ASSEMBLER, di scrivere in LM mediante il comando POKE x,y dato che sono indicati in decimale sia gli indirizzi che i contenuti degli indirizzi stessi.

I programmi che presento girano indifferentemente su PET vecchie, nuove e nuovissime ROM (BASIC 4), quindi per i modelli 3032 trasformati, CBM 4032 e CBM 8032.

Vediamo brevemente come si leggono i programmi scritti in Assembler; in questa puntata ve ne sono tre. Innanzitutto ad ognuno di essi si puo' assegnare un nome della lunghezza massima di 75 caratteri che compare poi in reverse in testa all'editing del programma stesso.

Seguono le Labels che possiedono un significato particolare nel corso del programma e che altro non sono se non nomi di fantasia da noi assegnati a punti del programma che consentono di rintracciare facilmente alcuni indirizzi o parti del programma stesso.

Tutti i programmi in Assembler terminano con un messaggio in reverse "PROGRAMMA SINTATTICAMENTE CORRETTO" che indica l'avvenuto rispetto, da parte nostra, delle norme da seguire nella stesura del programma.

Come si puo' notare non e' necessario indicare quasi mai gli indirizzi assoluti: mediante le Labels possiamo individuare punti di programma senza perdere tempo a calcolare gli indirizzi; e' il caso, tra gli altri, del BNE +LOOP della figura 2 o del JRS TRASFERISCI della figura 3 grazie ai quali il programma ASSEMBLER ci allevia della fatica di individuare gli indirizzi o di calcolare il "salto" o di dover riscrivere tutto il programma LM nel caso decidessimo di inserire una nuova istruzione. Si sa infatti che, in quest'ultimo caso, dopo l'inserimento della nuova istruzione e' quasi sempre necessario ricalcolare molti indirizzi di istruzioni a due o tre bytes ed alcuni indirizzamneti indiretti.

Consideriamo per esempio il programma di figura 2: quando il programma ASSEMBLER incontra il comando:

```
28672 = BLOCCO
```

automaticamente assegna alla Label BLOCCO il valore 28672 con il vantaggio che quando in seguito vorremo indicare tale indirizzo, sara' sufficiente chiamarlo con il nome BLOCCO anziche' con il suo indirizzo esadecimale o decimale (vedi appunto istruzione LDAX ;BLOCCO).

Un altro significativo esempio e' dato dal BNE +LOOP che, nel caso in cui il risultato della operazione in questione non sia uguale a zero, fa continuare il programma dal punto indicato precedentemente con il nome di fantasia LOOP.

Come si nota non e' stato necessario calcolare l'entita' del salto ma e' stato sufficiente trascrivere il nome LOOP nel punto in cui desideriamo che il salto avvenga: a rintracciare tale punto ed a calcolare il salto ci pensa il programma ASSEMBLER analogamente come il BASIC calcola il numero di righe e di istruzioni da saltare quando incontra i comandi di GOTO o GOSUB.

Tra gli altri vantaggi, vedi figura 3, e' possibile inserire dei commenti che, analogamente ai REM noti del BASIC, aiutano a rendere piu' comprensibile il programma stesso.

Da notare ancora che se noi assegnamo la Label PARTENZA alla locazione 7191, tale valore, per comodita', viene tradotto in esadecimale e trascritto in alto al fianco della Label.

Ancora: con istruzione del tipo INC ;INIZPRIM+1 e' possibile elaborare (nel caso specifico: incrementare) il contenuto di locazione di memoria non espressamente indicate, ma che "sono distanti" alcuni bytes dagli indirizzi dalle Label specificate.

Nel caso di INC ;INIZPRIM+1 verra' indicato il contenuto della locazione successiva a quella indicata precedentemente con INIZPRIM; analogamente sono accettate istruzioni del tipo: INC BLOCCO+4: a rintracciare il giusto indirizzo ci pensa naturalmente l'ASSEMBLER.

Il lettore che abbia seguito le due puntate precedenti e' in grado di caricare l'accumulatore con il contenuto di una qualsiasi locazione di memoria.

Studieremo questa volta un nuovo modo di indicare l'indirizzo che ci interessa utilizzando il registro indice X.

Tale registro ad 8 bit e' contenuto all'interno della CPU stessa e puo' essere utilizzato per inserire dei valori senza modificare minimamente il contenuto dell'accumulatore.

Vediamo alcune istruzioni che lo interessano:

LDX# Codice operativo (O.C.): A2. LDX# #2 (O.P. A2 02): Trascrive in X il valore 2.

LDX. .2 (O.P. A6 02): Carica in X il contenuto della locazione 2 della pagina zero.

INX (O.P. E8): Incrementa il valore corrente di X. E' da ricordare che se ad esempio il valore corrisponde a A1 questo passa ad A2, mentre se e' FF questo passa a 00.

DEX (O.P. CA): Come per INX che incrementa, DEX esegue l'azione di decrementare il valore di X.

TAX (O.P. AA): Trasferisce il valore corrente dell'accumulatore nel registro X.

TXA (O.P. 8A): Trasferisce il valore di X nell'accumulatore.

Da questa prima panoramica si nota subito che un vantaggio e' rappresentato dall'avere, in un certo senso, di un secondo accumulatore su cui operare senza disturbare il primo, al quale siamo abituati fino ad ora.

C'e' da dire che esiste un secondo vantaggio derivante dal poter utilizzare il registro X come riferimento per la ricerca di un indirizzo. Consideriamo, infatti, la nuova istruzione LDAX il cui codice operativo e' BD, (istruzione a tre bytes).

```
LDAX ;$8000
8D 00 80 in LM
```

Questa istruzione carica in accumulatore il contenuto della locazione il cui indirizzo e' rappresentato dai due bytes che seguono 8D (cioe' 80 00) ai quali e' sommato il valore corrente di X.

```
LDX# #2       LM: A2 02
LDAX ;$8000 LM: BD 00 80
```

In accumulatore sara' depositato il valore della locazione il cui indirizzo e' rappresentato dalla somma di 8000+02, vale quindi a dire 8002. Otterremo in definitiva il medesimo effetto di LDA ;$8002.

Una istruzione analoga e' anche il "deposito" di un dato in un indirizzo indicizzato come nell'esempio precedente.

```
STAX ;$8000   O.P.  9D 00 80
```

In effetti il medesimo risultato e' ottenibile utilizzando le istruzioni che gia' conosciamo. Vediamo allora la differenza tra i due metodi.

Esercizio: Trascrivere i contenuti di un blocco di memoria da 83C0 a 83E8 (decimale: da 33748 a 33768, che corrisponde esattamente all'ultima riga di schermo) a un nuovo blocco da 8000 a 8028, che corrisponde proprio alla prima riga di schermo.
Esaminiamo in un primo momento lo schema relativo al metodo noto (figura 1):
La prima parte (da 033A a 034E) costituisce la necessaria inizializzazione del programma (vedremo piu' avanti il perche').
L'istruzione che si trova in 034F carica la prima locazione e la deposita all'inizio del secondo blocco; successivamente si incrementano i bytes meno significativi mediante le istruzioni EE xx xx e si esamina il contenuto di X per sapere se si e' giunti al valore impostato 28 (decimale 40).
L'inizializzazione e' dovuta al fatto che se noi vogliamo eseguire una secinda volta senza ripristinare i valori iniziali (8000 e 83C0) gli indirizzi dei due blocchi si troveranno al punto in cui sono stati lasciati dopo la prima esecuzione, cioe' a 8028 e 83E8; non solo, ma il confronto (C9 28)

```
TRASFERIMENTO BLOCCO COL METODO NOTO
=BLOCCO
=SCHERMO
=LOOP
=PRELIEVO
=DEPOSITO

33728                    C33728
33728                          =BLOCCO
32768                    C32768
32768                          =SCHERMO
826                      C826
826   169                      LDA#
827   128                         $80
828   141                      STA
829    84  830      3             ;DEPOSITO+1
831   169                      LDA#
832   131                         $83
833   141                      STA
834    81  835      3             ;PRELIEVO+1
836   169                      LDA#
837   192                         $C0
838   141                      STA
839    80  840      3             ;PRELIEVO
841   169                      LDA#
842     0                         #0
843   141                      STA
844    83  845      3             ;DEPOSITO
846   170                      TAX
847                            =LOOP
847   173                      LDA
848                            =PRELIEVO
848   192  849    131             ;BLOCCO
850   141                      STA
851                            =DEPOSITO
851     0  852    128             ;SCHERMO
853   238                      INC
854    80  855      3             ;PRELIEVO
856   238                      INC
857    83  858      3             ;DEPOSITO
859   232                      INX
860   224                      CPX#
861    40                         #40
862   208                      BNE
863   239                         +LOOP
864    96                      RTS
PROGRAMMA SINTATTICAMENTE CORRETTO
```

```
033A A9 80 8D 54 03 A9 83 8D
0342 51 03 A9 C0 8D 50 03 A9
034A 00 8D 53 03 AA AD C0 83
0352 8D 00 80 EE 50 03 EE 53
035A 03 E8 E0 28 D0 EF 60 00
```

Figura 1

```
      TRASFERIMENTO INDICIZZATO
=BLOCCO
=SCHERMO
=LOOP


33728              033728
33728                     =BLOCCO
32768              $8000
32768                     =SCHERMO
  826              826
  826  162                LDX#
  827   40                    #40
  828                     =LOOP
  828  189                LDAX
  829  192  830  131          ;BLOCCO
  831  157                STAX
  832    0  833  128          ;SCHERMO
  834  202                DEX
  835  208                BNE
  836  247                    +LOOP
  837   96                RTS
 PROGRAMMA SINTATTICAMENTE CORRETTO
```

```
033A A2 28 BD C0 83 9D 00 80
0342 CA D0 F7 60 00 00 00 00
```

Figura 2

consente 256 incrementi prima di tornare al BASIC. In oltre siamo costretti a ricorrere al registro X per avere un contatore su cui operare.

Esaminiamo ora il programma in LM utilizzando la tecnica degli indirizzi indicizzati. (figura 2).

L'inizializzazione non e' piu' necessaria, il programma e' piu' snello, piu' comprensibile e oltretutto piu' veloce. Provate ad immaginare come si complicherebbe, questo programma, se non si ricorresse all'indirizzamento indicizzato nel caso in cui sia necessario considerare il carry per la parte alta degli indirizzi!

Quale esercizio, il lettore, puo' ora studiare in un primo momento il funzionamento dell'ultimo programma presentato nella precedente puntata, e quindi in seguito modificare quello di figura 3 di POCKET PET n.1, utilizzando la tecnica appena appresa.

Vediamo ora in che modo si possa trasferire dei blocchi con piu' di 256 locazioni di memoria.

Innazitutto c'e' da notare che le istruzioni indicizzate tengono conto, automaticamente, del CARRY, quando necessario. Per esempio utilizzando le seguenti due istruzioni, in accumulatore verra' riportato il valore della locazione presente all'indirizzo 7BFE+05, cioe' 7C04.

```
..........
LDX# #5
STAX,;7BFE
..........
```

Una routine che consente di trasferire un blocco di 1000 locazioni di una parte qualunque di memoria nella memoria di schermo in un modo pressoche' istantaneo puo' essere quella di figura 3 che commentiamo brevemente.

Come prima cosa e' da tenere presente che poiche' lo schermo del PET e' formato da 1000 locazioni di memoria RAM, effettueremo tre volte un trasferimento di 256 locazioni piu' uno di 1000-(256*3), cioe' 232 che in esadecimale corrisponde a E8.

Pertanto all'inizio caricheremo Y con il valore 3 e per tre volte eseguiremo 256 volte un salto alla subroutine TRASFERISCI (20 6B 03) che consente di prelevare e depositare i dati avendo come riferimento X.

JSR (O.P. 20 xx yy) e' una istruzione a tre bytes che ci permette di saltare alla subroutine il cui indirizzo e' rappresentato dai due bytes seguenti. Il lettore gia' conosce l'RTS, istruzione implicita ad un byte (O.P. 60).

Effettuato il trasferimento dei tre blocchi, ora ci rimane quello da 232 bytes. Le prime istruzioni inizializzano le locazioni INIZPRIM+1 e INIZSEC+1.
:ln2;Potremo modificare da BASIC con tre POKE i contenuti delle locazioni 827 832 e 837 in modo tale da poter trasferire qualsiasi gruppo di 1000 locazioni in altre parti qualsiasi di memoria RAM.

```
      PAGINE DI MEMORIA                                   =PARTENZA          =ULTIMO
                                                          =ARRIVO            =TRASFERISCI
 7168                       $1C00                         =BLOCCO            =INIZPRIM
 7168                            =PARTENZA                =LOOP              =INIZSEC
32768                       $8000
32768                            =ARRIVO
  826                       $826

                                                   ;INIZIALIZZA GLI INDIRIZZI
                                                   ;INIZIALI SCELTI 8000 1C00

  826  169                  LDA#
  827  128                      $80
  828  141                  STA
  829  112  830    3            ;INIZSEC+1
  831  169                  LDA#
  832   28                      $1C                                      Figura 3
  833  141                  STA
  834  109  835    3            ;INIZPRIM+1
  836  169                  LDA#
  837    0                      #0
  838  141                  STA
  839  111  840    3            ;INIZSEC
  841  141                  STA
  842  108  843    3            ;INIZPRIM
                                                   ;ESEGUE 3 VOLTE LOOP

  844  160                  LDY#
  845    3                      #3
  846                            =BLOCCO
  846  162                  LDX#
  847    0                      #0
  848                            =LOOP
  848   32                  JSR
  849  107  850    3            ;TRASFERISCI
  851  232                  INX
                                                   ;ESEGUE 256 TRASFERIMENTI
                                                   ;DI CARATTERI

  852  208                  BNE
  853  250                      +LOOP
                                                   ;INCREMENTA [illegible] DEGLI
                                                   ;INDIRIZZI INTERESSATI

  854  238                  INC
  855  109  856    3            ;INIZPRIM+1
  857  238                  INC
  858  112  859    3            ;INIZSEC+1
  860  136                  DEY
  861  208                  BNE
  862  239                      +BLOCCO
  863   32                  JSR
  864  107  865    3            ;TRASFERISCI
  866  162                  LDX#
  867  232                      $E8
  868                            =ULTIMO
                                                   ;ULTIMO BLOCCO DI 232 CARATTERI

  868   32                  JSR
  869  107  870    3            ;TRASFERISCI
  871  202                  DEX
  872  208                  BNE
  873  250                      +ULTIMO                033A A9 80 8D 70 03 A9 1C 8D
  874   96                  RTS                        0342 6D 03 A9 00 8D 6F 03 8D
  875                            =TRASFERISCI          034A 6C 03 A0 03 A2 00 20 6B
  875  189                  LDAX                       0352 03 E8 D0 FA EE 6D 03 EE
  876                            =INIZPRIM             035A 70 03 88 D0 EF 20 6B 03
  876    0  877   28            ;PARTENZA              0362 A2 E8 20 6B 03 CA D0 FA
  878  157                  STAX                       036A 60 BD 00 1C 9D 00 80 60
  879                            =INIZSEC
  879    0  880  128            ;ARRIVO
  881   96                  RTS
 PROGRAMMA SINTATTICAMENTE CORRETTO
```

NATALE POCKET PET

di Gloriano Rossi

Una redazione di un giornale o di una rivista risulta essere sempre un luogo piene di carte, di fascicoli, matite e penne, pellicole e gente che va' e che viene.
Anche la redazione di Pocket PET non fa eccezione a questa regola.
Per rendere un po' rilassante questo ambiente cosi' frenetico e caotico, abbiamo pensato che in occasione delle feste natalizie e di fine d'anno di inserire una nota di allegria e di pace nello spazio redazionale:
un alberello di Natale.
Un albero un po' diverso da quello che viene normalmente allestito negli uffici di persone "normali".
Infatti in redazione di Pocket PET non ci si poteva accontentare del classico abete corredato di festoncini e di lucette intermittenti all'unisono, e... quindi:
PET alla mano, ed ecco che nasce un programmino che ad analisi compiuta doveva avere queste semplici prerogative:

- disegnare un piccolo abete stilizzato sullo schermo.
- scrivere la causale "Pocket PET Natale 1981" con un sistema ed una grafica adatta alle circostanze.
- simulare l'accensione e lo spegnimento di ipotetiche lampadine sul video.
- dato che in un Natale che si rispetti nevica, si doveva far nevicare.
- generare tramite la porta di uscita CB2 della USER-PORT (piedino M) e la massa (piedino N) un segnale che opportunamente amplificato diffondesse per "l'aire" i classici motivetti che si ascoltano generamente in queste occasioni.

E.... 'dulcis in fundo':

- tramite una semplice interfaccia e per mezzo delle uscite che vanno da PA0 a PA7 della USER-PORT poter pilotare otto serie di lampadine, vere questa volta, installate su un altrettanto vero albero di Natale.

Per riassumere si puo' dire che la realizzazione completa richiede essenzialmente l'applicazione alla USER-PORT di un semplice amplificatorino di bassa frequenza e la costruzione di una altrettanto semplice interfaccia fra il PET e la sua nuova perfiferica costituita dal modello "abete con luci".
L'attualizzazione di tutti questi passi non e' tassativo al fine del buon funzionamento del programma.
Il programma gira anche se non viene applicata ne l'interfaccia PET-Albero e neppure se si omette il piccolo amplificatore.
Tutto sta quindi alla scelta del lettore quale potra' essere lo stadio della realizzazione da raggiungere.

Il programma.

Prima di parlare della parte Hardware vediamo di analizzare quel mezzo che la redazione utilizza per portare gli auguri di Pocket PET nelle vostre case.
Questo programma e' sostanzialmente diviso in parti ben distinte che possiamo analizzare una ad una con la

solita tecnica di:

REMarks.

1-6 In queste poche righe e' situato il programmino/routine scritto in linguaggio macchina che ci permettera' di ottenere l'effetto nevicata a suon di musica. Questa routine e' stata scritta in Assembler in quanto per ottenere il medesimo effetto in BASIC si sarebbe pregiudicata l'esecuzione dei pezzi musicali. La routine e' applicabile a tutti i tipi di PET Commodore con l'esclusione del modello 8032 e del VIC20.
L'interprete BASIC rispetto al linguaggio macchina diretto, perde moltissimo tempo per eseguire le varie funzioni richiamate. In Assembler invece i comandi sono talmente veloci che la routine proposta, nonostante la completezza dell'effetto ottenuto, viene eseguita in un batter d'occhio e, soppratuttto, senza implicare un rallentamento della musica e degli effetti che il programma e' in grado di fornire.
La zona di memoria interessata da questa routine e' quella del buffer della seconda unita' a cassetta magnetica ed il sistema usato per scrivere i comandi in questa zona di memoria e' quello di eseguire delle POKE in successione con i valori decimali dei vari comandi ed istruzioni di Assembler.
Dalla locazione 830 fino alla locazione 937 e' quindi situata la "nevicata".

7 La routine per girare regolarmente utilizza anche un'altra zona della memoria RAM del sistema, sistuata oltre il programma BASIC.
In questa riga si inizializza con valori casuali (RANDOM) questa area di lavoro.

8-37 In queste semplici righe si attualizza sullo schermo le scritte coreografiche "Pocket PET

Indirizzo num	Indirizzo esa	dati	mnemonico
830	033E	A2 00	LDXIM 0
832	0340	AC 3B 03	LDY 827
835	0343	B9 00 1F	LDAY 7936
838	0346	85 62	STAZ 98
840	0348	B9 01 1F	LDAY 7937
843	034B	85 63	STAZ 99
845	034D	A1 62	LDAIX 98
847	034F	C9 2E	CMPIM 46
849	0351	D0 04	BNE 4
851	0353	A9 20	LDAIM 32
853	0355	81 62	STAIX 98
855	0357	B9 00 1F	LDAY 7936
858	035A	38	SEC
859	035B	E9 29	SBCIM 41
861	035D	99 00 1F	STAY 7936
864	0360	B9 01 1F	LDAY 7937
867	0363	E9 00	SBCIM 0
869	0365	99 01 1F	STAY 7937
872	0368	B9 00 1F	LDAY 7936
875	036B	18	CLC
876	036C	6D 3A 03	ADC 826
879	036F	99 00 1F	STAY 7936
882	0372	B9 01 1F	LDAY 7937
885	0375	69 00	ADCIM 0
887	0377	99 01 1F	STAY 7937
890	037A	C9 84	CMPIM 132
892	037C	D0 05	BNE 5
894	037E	A9 80	LDAIM 120
896	0380	99 01 1F	STAY 7937
899	0383	09 7F	CMPIM 127
901	0385	D0 05	BNE 5
903	0387	A9 83	LDAIM 131
905	0389	99 01 1F	STAY 7937
908	038C	B9 00 1F	LDAY 7936
911	038F	85 62	STAZ 98
913	0391	B9 01 1F	LDAY 7937
916	0394	85 63	STAZ 99
918	0396	A1 62	LDAIX 98
920	0398	C9 20	CMPIM 32
922	039A	D0 04	BNE 4
924	039C	A9 2E	LDAIM 46
926	039E	81 62	STAIX 98
928	03A0	88	DEY
929	03A1	88	DEY
930	03A2	C0 FE	CPYIM 254
932	03A4	F0 03	BEQ 3
934	03A6	4C 43 03	JMP 835
937	03A9	60	RTS

Nel listato del programma in BASIC esiste una parte in linguaggio macchina. ecco in versione Assembler quella routine che ci permette di generare e far cadere la neve.

Natale in redazione di

Pocket PET

```
1 FOR J=830 TO 937:READ X:POKE J,X:NEXT
2 DATA 162,0,172,59,3,185,0,31,133,98,185,1,31,133,99,161,98,201,46,208,4,169
3 DATA 32,129,98,185,0,31,56,233,41,153,0,31,185,1,31,233,0,153,1,31,185,0
4 DATA 31,24,109,58,3,153,0,31,185,1,31,105,0,153,1,31,201,132,208,5,169,128
5 DATA 153,1,31,201,127,208,5,169,131,153,1,31,185,0,31,133,98,185,1,31,133
6 DATA 99,161,98,201,32,208,4,169,46,129,98,136,136,192,254,240,3,76,67,3,96

7 FOR I=7936 TO 8191 STEP 2:POKE I,RND(5)*256:POKE I+1,RND(5)*4+128:NEXT
8 PRINT "[CLR]"
9 PRINT "┌────┐ ┌────┐
10 PRINT "└┐  ┌┘ └┐  ┌┘ POCKET PET
11 PRINT " │  └───┘  │
12 PRINT " │  ┌───┐  │
13 PRINT "┌┘  └┐ ┌┘  └┐
14 PRINT "└────┘ └────┘
15 I=35
16 PRINT "[HOME]" TAB(I)"┌─╮"
17 PRINT TAB(I)"│ │"
18 PRINT TAB(I)"│ │"
19 PRINT TAB(I)"╭─╮"
20 PRINT TAB(I)"├─┤"
21 PRINT TAB(I)"╱ ╲"
22 PRINT TAB(I)"╭┬╮"
23 PRINT TAB(I)" │ "
24 PRINT TAB(I)" │ "
25 PRINT TAB(I)"╭─╮"
26 PRINT TAB(I)"├─┤"
27 PRINT TAB(I)"╱ ╲"
28 PRINT TAB(I)"╲  "
29 PRINT TAB(I)"│  "
30 PRINT TAB(I)"└─╯"
31 PRINT TAB(I)"┌─╮"
32 PRINT TAB(I)"├─ "
33 PRINT TAB(I)"└─╯"
34 I=27
35 PRINT "┬╮┌─ ┌┬╮" TAB(I)" ┐╭─╮╭─╮ ┐"
36 PRINT "├╯├─  │ " TAB(I)" │╰─┤├─┤ │"
37 PRINT "╵ └─╮ ┴ " TAB(I)" ┴╰─╯╰─╯ ┴"
38 DIM NT(24),BP(24),SV(5),RE(5)
39 Q0=142:IF PEEK(50000)THEN Q0=60
40 B=255:P=2↑(1/12)
41 FOR I=1 TO 24:NT(I)=INT(B+.5):B=B/P:NEXT I
42 NT(0)=0
43 TB=20:T$="":PRINT "[HOME]"
44 PRINT TAB(TB);"[UP][RVS]◤◥[OFF][ 2 LEFT][DOWN]◥◤"
45 FOR I=1 TO 9:T$=T$+CHR$(34)
46 PRINT TAB(TB-I);" ╱";TAB(TB+I);"╲ "
47 PRINT TAB(TB-I);"╱ ";TAB(TB+I);" ╲"
48 NEXT I:PRINT TAB(TB-I+1);T$;"▒▒";T$
49 PRINT TAB(TB);"▒▒"
50 PRINT TAB(TB-2)"◥[RVS]    [OFF]◤"
51 PRINT TAB(TB-1)"◥[RVS]  [OFF]◤"
52 PRINT "[HOME]"
53 FOR I=0 TO 24
54 T=INT(.5+SQR(.25+2*INT(RND(1)*45)))
55 R=2*T+1:IF RND(1)>.5 THEN R=R+1
56 IF R>19 THEN 54
57 C=INT(RND(1)*2*T)-T+1
58 BP=32768+40*R+C+TB:IF PEEK(BP)<>32 THEN 54
59 BP(I)=BP:POKE BP,96
60 NEXT I
```

DATA	Indice Tabella	Valore POKE	Nota Corrisp.
A	1	255	DO b
B	2	241	DO
C	3	227	DO #
D	4	214	RE
E	5	202	RE #
F	6	191	MI
G	7	180	FA
H	8	170	FA #
I	9	161	SOL
J	10	152	SOL#
K	11	143	LA
L	12	135	LA #
M	13	127	SI
N	14	120	DO
O	15	114	DO #
P	16	107	RE
Q	17	101	RE #
R	18	96	MI
S	19	90	FA
T	20	85	FA #
U	21	80	SOL
V	22	76	SOL#
W	23	72	LA
X	24	68	LA #

Tabellina relativa alle relazioni fra i dati riportati nei DATA del programma ed i valori delle POKE da eseguire al fine di ottenere le giuste note musicali richeste nei brani musicali.

```
61 FOR I=0 TO 24:POKE BP(I),32:NEXT I
62 FOR I=0 TO 5:RE(I)=PEEK(Q0+I):NEXT I
63 POKE 59467,16:POKE 59466,15:POKE 59464,0:PN=-1
64 READ N$,D:POKE 827,100:POKE 826,81
65 IF D<=0 THEN 77
66 NT=ASC(N$)-64:BL=INT(RND(1)*25):PC=NT(NT)
67 BP=BP(BL):NB=113-PEEK(BP)
68 IF TI<TM THEN 68
69 IF PN=PC THEN POKE 59464,0
70 POKE 59464,PC:POKE BP,NB
71 POKE 59459,INT(255*RND(1)):SYS 830
72 PN=PC
73 TM=TI+D*4
74 GET T$:IF T$="" THEN 64
75 POKE 59464,0:POKE 59466,0:POKE 59467,0
76 END
77 D=-D
78 IF N$=">" THEN FOR Z=0 TO 5:SV(Z)=PEEK(Q0+Z):NEXT Z:RP=1:GOTO 64
79 IF N$="↑" AND RP<D THEN FOR Z=0 TO 5:POKE Q0+Z,SV(Z):NEXT Z:RP=RP+1:GOTO 64
80 IF N$="<" THEN FOR I=0 TO 5:POKE Q0+I,RE(I):NEXT I:GOTO 64
81 GOTO 64
82 REM ====== DECK THE HALLS ...
83 DATA>,0,O,7,M,2,L,4,J,4,H,4,J,4,L,4,H,4,J,2,L,2,M,2,J,2,L,6,J,2,H,4,G,4,H,8
84 DATA O,7,M,2,L,4,J,4,H,4,J,4,L,4,H,4,J,2,L,2,M,2,J,2,L,6,J,2,H,4,G,4,H,8
85 DATA J,7,L,2,M,4,J,4,L,7,M,2,O,4,J,4,L,2,N,2,O,4,Q,2,S,2,T,4,S,4,Q,4,O,8
86 DATA O,7,M,2,L,4,J,4,H,4,J,4,L,4,H,4,Q,2,Q,2,Q,2,Q,2,O,7,M,2,L,4,J,4,H,8,@,4
87 DATA↑,-3,@,32
88 REM ====== GOOD KING ...
89 DATA>,0,H,4,H,4,H,4,J,4,H,4,H,4,C,8,E,4,C,4,E,4,G,4,H,8,H,8
90 DATA H,4,H,4,H,4,J,4,H,4,H,4,C,8,E,4,C,4,E,4,G,4,H,8,H,8
91 DATA O,6,M,1,L,4,J,4,L,4,J,4,H,8,E,4,C,4,E,4,G,4,H,8,H,8
92 DATA C,4,C,4,E,4,G,4,H,4,H,4,J,8,O,4,M,4,L,4,J,4,H,10,M,10,H,10,@,8
93 DATA↑,-3,@,32
94 REM ====== JINGLE BELLS
95 DATA>,0,J,4,J,4,J,8,J,4,J,4,J,8,J,4,M,4,F,6,H,1,J,16
96 DATA K,4,K,4,K,6,K,2,K,4,J,4,J,6,J,2,J,4,H,4,H,4,J,4,H,9,M,8
97 DATA J,4,J,4,J,8,J,4,J,4,J,8,J,4,M,4,F,6,H,1,J,16
98 DATA K,4,K,4,K,6,K,2,K,4,J,4,J,6,J,2,M,4,M,4,K,4,H,4,F,16
99 DATA A,4,J,4,H,4,F,4,A,12,A,3,A,1,A,4,J,4,H,4,F,4,C,16
100 DATA C,4,K,4,J,4,H,4,E,16,M,4,M,4,K,4,H,4,J,16
101 DATA A,4,J,4,H,4,F,4,A,16,A,4,J,4,H,4,F,4,C,16
102 DATA C,4,K,4,J,4,H,4,M,4,M,4,M,4,M,4,O,4,M,4,K,4,H,4,F,16,↑,-2
103 DATA J,4,J,4,J,8,J,4,J,4,J,8,J,4,M,4,F,6,H,1,J,16
104 DATA K,4,K,4,K,6,K,2,K,4,J,4,J,6,J,2,J,4,H,4,H,4,J,4,H,9,M,8
105 DATA J,4,J,4,J,8,J,4,J,4,J,8,J,4,M,4,F,6,H,1,J,16
106 DATA K,4,K,4,K,6,K,2,K,4,J,4,J,6,J,2,M,4,M,4,K,4,H,4,F,16,@,32
107 REM ====== RUDOLPH THE
108 DATA>,0,H,2,J,3,H,1.5,E,3,M,3,J,3,H,10,H,2,J,1,H,2,J,1,H,3,M,3,L,13
109 DATA F,2,H,3,F,1.5,C,3,L,3,J,3,H,10,H,2,J,1,H,2,J,1,H,3,J,3,E,13
110 DATA H,2,J,3,H,1.5,E,3,M,3,J,3,H,10,H,2,J,1,H,2,J,1,H,3,M,3,L,13
111 DATA F,2,H,3,F,1.5,C,3,L,3,J,3,H,10,H,2,J,1,H,2,J,1,H,3,O,3,M,13
112 DATA J,3,J,3,M,3,J,3,H,3,E,3,H,6,F,3,J,3,H,3,F,3,E,12,C,3,C,3,H,3,H,3
113 DATA L,3,L,3,O,6,M,3,M,3,L,3,J,3,H,3,F,3,C,6
114 DATA H,2,J,3,H,1.5,E,3,M,3,J,3,H,10,H,2,J,1,H,2,J,1,H,3,M,3,L,13
115 DATA F,2,H,3,F,1.5,C,3,L,3,J,3,H,10,H,2,J,1,H,2,J,1,H,3,O,3,M,13,↑,-2,@,32
116 REM ====== SILENT NIGHT
117 DATA>,0,H,8,J,2,H,6,E,18,H,8,J,2,H,6,E,18,O,10,O,6,L,16,M,10,M,6,H,16
118 DATA J,10,J,6,M,8,L,2,J,6,H,8,J,2,H,6,E,16
119 DATA J,10,J,6,M,8,L,2,J,6,H,8,J,2,H,6,E,16
120 DATA O,10,O,6,R,8,O,2,L,6,M,16,Q,16,M,5,H,5,E,5,H,8,F,2,C,6,A,18,@,18
121 DATA H,8,J,2,H,6,E,18,H,8,J,2,H,6,E,18,O,10,O,6,L,16,M,10,M,6,H,16
122 DATA J,10,J,6,M,8,L,2,J,6,H,8,J,2,H,6,E,16
123 DATA J,10,J,6,M,8,L,2,J,6,H,8,J,2,H,6,E,16
124 DATA O,10,O,6,R,8,O,2,L,6,M,16,Q,16,M,5,H,5,E,5,H,8,F,2,C,6,A,18,↑,-2,@,32

125 DATA<,0
```

Natale 1981".
Questi caratteri verranno posti sullo schermo in posizioni esteticamente studiate.

38 Dimensionamento di alcune tabelle che vengono utilizzate nel programma.

39 Per togliere ogni possibile dubbio si puo' tranquillamente dire che questo programma puo' girare sia con PET vecchie che nuove ROM, nonche' con CBM serie 4000. Infatti in questa riga il sistema si accorge di quale edizione di ROM sta' utilizzando ed a causa di cio' adegua il valore della variabile Q0. Questa variabile interessa alcuni dei pointer di pagina zero. (per i neofiti si puo' dire che la pagina zero non e' altro che la prima zona di memoria RAM del sistema e che e' riservata e gestita dal microprocessore 6502)

40-42 Una delle routine di questo programma veramente interessante e' proprio scritta in queste tre righe. Quattro istruzioni in croce ed ecco due ottave musicali nei rispettivi numeri corrispondenti alle POKE interessate. I valori ricavati dalla formula (per convenzione universale si divide la frequenza della nota precedente per due elevato ad un dodicesimo per ottenere la nota seguente) sono posti in una tabellina chiamata NT (da NoTa).
Il primo elemento della tabella, quello indicizzato con il numero zero, viene inizializzato con il valore zero, numero che significhera' assenza di nota.

43-52 Ecco un altro gruppo di istruzioni alquanto semplici che avranno la sola funzione di disegnare l'albero di Natale completo di puntale e di vaso di base.

53-61 Con questa routine si identificano quelle posizioni della memoria di schermo (in pratica i punti del video) verranno interessati alla futura accensione e spegnimento delle pseudo lampadine, 24 in totale.
Le istruzioni terminano riportando il valore corrispondente allo spazio la' dove per necessita' di routine e' stato forzato il valore numerico 96.

62 Memorizzazione in tabella dei valori interessati delle locazioni di memoria di pagina zero.

63 Con questi comandi di POKE si apre la funzione di musica all'interno del PET senza pero' generare alcuna nota. Cosa abbiamo fatto in sintesi eseguendo questi comandi?
E' presto detto : con la prima POKE si da il vero comando di apertura del modo musica; con il secondo comando si sono scelte le due ottave musicali che il PET puo' generare senza dover cambiare il contenuto della posizione di memoria contrassegnata con 59466.
Cosi' per curiosita' si potra' provare con le due ottave seguenti e con le due seguenti ancora, variando la cifra 15 con 51 o con 85.

64-81 Questo e' veramente il cuore del programma. In questa zona avviene proprio di tutto: dalla generazione dei suoni, alla accensione e spegnimento delle pseudo lampadine, al pilotaggio dell'interfaccia che provvedera' a far lampeggiare le vere lampadine del vostro albero di Natale ed infine la nevicata. Il punto focale e' situato esattamente in riga 70 e in riga 71.
I comandi speciali sui quali fa perno tutto il programma sono:

POKE 59464,PC genera la nota individuata con le istruzioni precedenti

POKE BP,NB spegne od accende la pseudo lampadina individuata in maniera RANDOM all'interno dell'albero

POKE 59459,INT(255*RND(1)) genera un numero casuale che interessa direttamente la USER-PORT da PA0 a PA7

SYS (830) fa scendere la neve, in sintesi esegue la routine in linguaggio macchina

In riga 74 se nessun tasto viene premuto si continua ad eseguire il brano/i musicali con cio' che ne segue. In caso contrario, come un buon direttore d'orchestra che riponendo la bacchetta chiude l'esecuzione di un concerto, cosi' il nostro PET pone a zero quelle tre locazioni di memoria interessate alla generazione della musica. Questo passo e' obbligatorio in quanto il circuito integrato 6522 VIA (versatile interfaccia adattatore) interessa anche la gestione di entrambe le unita' a cassetta magnetica. Lasciando aperte quelle porte, inevitabilmente, si ottengono dei risultati negativi in fase di LOAD o di SAVE.

82-125 Queste righe comprendono tutti i valori delle note (lettere), delle pause o mantenimenti (numeri) e dei comandi di riconoscimento di fine testo musicale. Questi ultimi ci permettono di influenzare quelle locazioni di pagina zero interessate alla gestione dei pointers (righe 78-79-80) permettendo cosi' l'esecuzione in continuo dei cinque brani musicali.

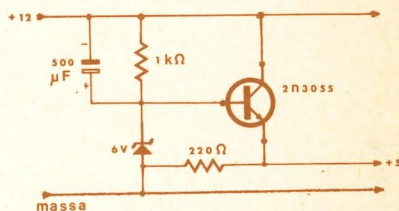
L'Hardware.

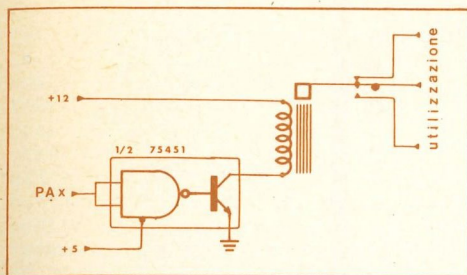
E' inutile dire che l'amplificatore di bassa frequenza andra' collegato alle prese M ed N della USER-PORT. La generazione delle note sonore avvengono proprio alla porta CB2 (M) e la massa (N).
Veniamo subito quindi alla parte piu' interessante della realizzazione, quella cioe' che ci permettera' di comandare le luci dell'albero di natale tramite il nostro PET.

L'interfaccia che mi accingo a descrivere potra' essere impiegata anche per altre realizzazioni quali ad esempio comandare qualsiasi circuito elettrico che necessita un controllo esterno preordinato.
Non esiste quindi la limitazione all'uso specifico natalizio, ma la versatilita' dell'utilizzazione e' lasciata alla mente estrosa del lettore.

Il circuito.

Il circuito elettronico e' alquanto semplice.
Un normale alimentatore a tensione continua che puo' erogare una tensione compresa fra i valori di 9 e 12 volt con 200/300 mA, sara' piu' che sufficente a fornire il voltaggio necessario a tutti i componenti.
Essenzialmente lo schema puo' essere suddiviso in tre parti ben distinte.

L'alimentazione. I circuiti integrati che vengono utilizzati richiedono una tensione di alimentazione di circa 5.1 volt stabilizzati, mentre la sezione relay necessita di una tensione di 9/12 volt.
Un transistore di potenza NPN tipo 2N3055, un diodo speciale zenner da 6 volt, due resistenze ed un condensatore elettrolittico opportunamente collegati fra di loro ci forniscono la bassa tensione di alimentazione necessaria ai circuiti integrati 75451.

Questo piccolo schema e' composto da componenti atti a ridurre e stabilizzare la tensione di alimentazione da 12/15 volt ai circa 5.5 volt necessari per il buon funzionamento degli integrati tipo 75451.

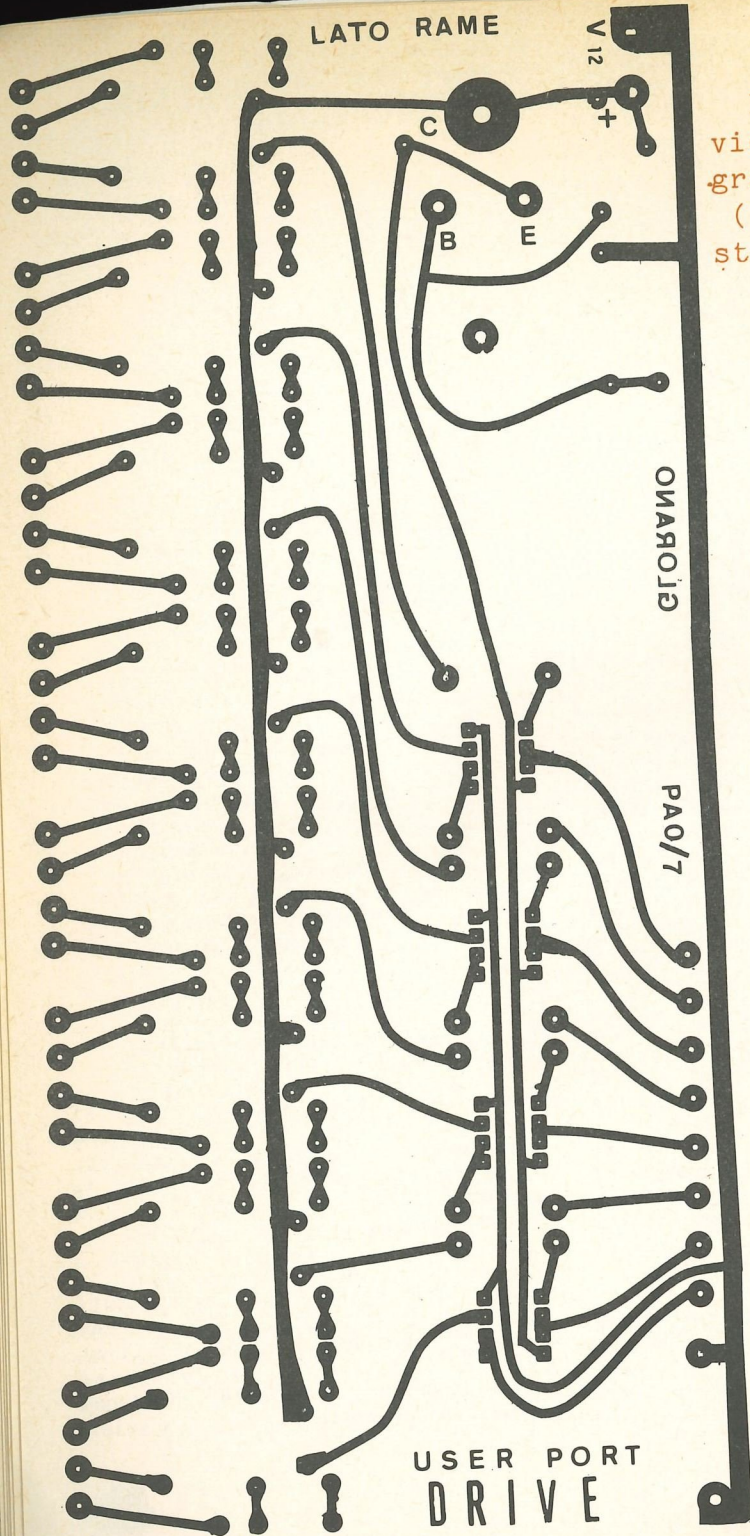
TTL Drive. La seconda parte dello schema e' costituita da quattro integrati della serie TTL (Transistor-Transisto-Logic). Questi integrati, di facilissima reperibilita', sono stati studiati appositamente per poter pilotare altri circuiti che utilizzano tensioni fino a 30 volt e che necessitano un fan-out elevato. Proprio per questo potere elevato di pilotaggio (fan-out) questo integrato viene comunemente utilizzato in grossi calcolatori per attualizzare le funzioni di rinfresco delle memorie dinamiche.

Schema elettrico di base di cui e' costituito il circuito di DRIVE. Questo schema e' stato duplicato, nella realizzazione pratica, otto volte in quanto otto sono le uscite della USER-PORT.

Molto in breve vengono riportate le caratteristiche elettriche di funzionamento, le connessioni interne ed i dati di ingombro del circuito integrato della serie TTL tipo 75451.

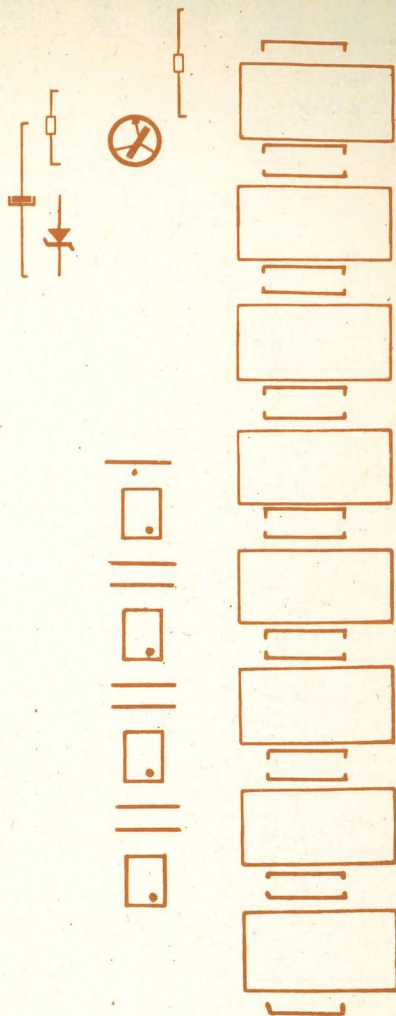
Osservando il disegno relativo alle connessioni interne del 75451, questi e' costituito da due funzioni uguali ognuna delle quali comprende una porta NAND a due ingressi ed un transistore che ha la vera e propria funzione di drive. Oltre che avere una funzione di pilota questo transistore e' un invertitore di stato, per questa ragione il circuito nel suo complesso logico e' un AND a due ingressi.

Allo schema non interessa l'esistenza di un doppio ingresso per cui l'uscita della USER-PORT sara' collegata direttamente ad entrambi gli ingressi del NAND.

Quando il PAx della USER-PORT cambiera' di stato il transistore sara' messo in condizione di condurre, si permettera' quindi il passaggio di corrente fra il colletore e l'emettitore.

Relay. Come gia' detto, ogni 75451 comprende due circuiti di drive ben distinti; ogni integrato allora potra' pilotare singolarmente due relay ai quali andranno applicate funzioni specifiche preordinate.

In questa figura
viene riportato in
grandezza naturale
(1:1) il circuito
stampato lato rame
del DRIVE.

Riassunto:
-PAx della USER-PORT del PET cambia di stao logico.
-La porta NAND riconosce questo cambiamento e fa si che:
-il transistore conduca e quindi:
-il relay viene eccitato in quanto il circuito e' chiuso verso massa.

Il circuito stampato proposto in queste pagine vede l'utilizzo di relay della SIEMENS tipo V23012 A0002-A001. Il costo di questi componenti non e' eccessivamente elevato, ma se questo fatto in realta' sussistesse od al contrario fosse richiesta una diversa caratteristica di capacita' fra i contatti, nessuno vieta di modificare il disegno delle piste di rame ed adattarle al tipo di relay prescelto. L'utilizzazione di un altro tipo di relay dovra' sottostare alla caratteristiche elettriche che il circuitino di DRIVE richiede. La bobina che costituisce parte dell'elettromagnete dovra' avere una tensione di eccitazione non superiore ai 12/15 volt ed non dovra' richedere per il suo funzionamento di una corrente maggiore ai 300 mA.
Nella figura che mostra l'esatta disposizione dei componenti si possono notare alcuni ponticelli.
I ponticelli, costituiti da comune cavetto di rame, che si trovano vicino agli integrati sono obbligatori, mentre quelli che notiamo vicino ai relay possono venire in parte omessi, tutto cio' in relazione alla destinazione pratica dell'interfaccia. Cosi' come e' stato suggerito nella disposizione dei componenti ed utilizzando quel tipo di relay, si ha una logica chiamata NA (normalmente aperto). Cio' vuole dire che quando si ha un cambiamento di stato alla USER-PORT si avra' la possibilita' di un circuito elettrico chiuso. Al contrario se i/il ponticelli fossero messi fra' la piazzolina in basso e quella mediana si avrebbe una logica NC (normalmente chiuso) e cosi' quando arriva il segnale, il circuito si apre e l'eventuale utilizzazione si arresterebbe.

Ecco come devono essere posti i vari componenti sulla basetta del circuito stampato. Notare i ponticelli eseguiti, in particolare quelli vicino ad i relay.

COMAL-80

Sulla rivista BIT n.22 e' apparsa una notizia riguardante il nuovo linguaggio per il PET, il:

C O M A L - 8 0

riportiamo integralmente il testo:

--*H*--

Nuovo linguaggio per il PET Commodore della Harden S.p.A.

La HARDEN SpA, distributrice in esclusiva del PET-COMMODORE distribuira' gratuitamente a coloro che ne facciano richiesta il COMAL-80.
Il COMAL-80 e' un linguaggio BASIC strutturato che unisce i vantaggi del Pascal alla semplicita' del BASIC.
E' stato originariamente studiato per scopi educativi, al fine di proporre un tipo di linguaggio di facile utilizzo, strutturato, e di alta leggibilita', si ritiene tuttavia che possa risultare estremamente utile anche in altri tipi di applicazione non esclusa quella professionale.
Il COMAL-80 (28k di codice oggetto) include, oltre agli operatori usuali del BASIC- come INPUT, PRINT, LET, ecc. -alcuni statments di tipo strutturale che consentono all'utente di trattare separatamente blocchi di istruzioni.
Ad esempio:
IF-ELSE-ENDIF, REPEAT-UNTIL, WHILE-ENDWHILE, CASE-WHEN-ENDCASE.

Vi sono inoltre alcune estensioni rispetto al BASIC come esempio la possibilita' di definire variabili lunghe fino a 16 caratteri. Le subroutine vengono richiamate per nome e non per numero ed ogni chiamata puo' includere il passaggio di parametri.
Il CBM-COMAL e' un progetto in via di sviluppo; e' gia' disponibile una versione detta "split version" che consente di lasciare un maggior spazio in RAM per l'utente, si sta inoltre studiando una versione anche per il VIC.

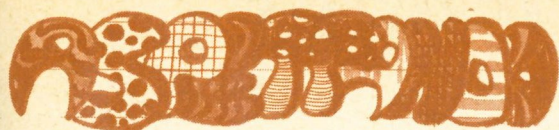
--*H*--

In questa sede la HARDEN vuole confermare la gratuita' del programma COMAL-80, escluso naturalmente il supporto, il costo della copia del manuale e delle spese di spedizione.
Occorre pero' ribadire che per soddisfare tutte le richieste e' necessario che ogni interessato faccia pervenire alcune informazioni basilari.

1) Nome e cognome
2) Indirizzo
3) Tipo di PET-CBM (4032 O 8032)
4) Tipo di Floppy disk (4040 O 8050)
5) Causale : richiesta COMAL-80

Per comunicare tutto cio' e sufficente anche una semplic cartolina postale ed in breve tempo con il solo onere del rimborso dell vive spese, che si aggirano intorn alle L.15000, potrete avere un nuov e potente linguaggio.

VIC 20

Aspettando il VIC ....

di
Stefano Miari

Eccolo finalmente! Questa "diabolica" tastiera, capace di soddisfare i piu' esigenti computeristi, e'giunta ora anche in Italia.
Un'anno fa era stata presentata la versione a caratteri giapponesi di questo prodotto della Commodore distribuito dalla Harden. E fu proprio allora che capii quanto e' importante oggi saper programmare ed usare un computer.
Io che scrivo, pur essendo privo di conoscenze di "ROM" "RAM" e "POKE", nell'accostarmi al mondo dell'informatica, sono rimasto stupefatto di fronte alle piccole dimensioni,alle possibilita' di espansione, al prezzo ed alla facilita' d'uso del "VIC 20".Per citare qualche caratteristica: ha la possibilita' di avere 16 colori ad alta risoluzione e otto di fondo, una memoria "RAM" espandibile fino a 32 Kbytes, tre toni contemporanei di suono e cosi' via di questo passo........
E' recentissima fra le altre cose una pubblicazione del "Gruppo Editoriale Jackson" sull'uso e le caratteristiche del "VIC 20", distribuita anche dalla Harden.Spa, e dai suoi rivenditori autorizzati. Questo libro, il cui titolo e': IMPARIAMO A PROGRAMMARE IN BASIC CON IL VIC 20, e' stato curato dalla professoressa Rita Bonelli.
La pubblicazione si prefigge innanzitutto di insegnare a programmare ai profani, ed e' scritto in modo semplice e comprensibile.
Purtroppo e' tutto quello che si puo' avere fino ad ora in lingua italiana.
Nell'attesa, mi ero premurato di avere dettagliate informazioni sui corsi di programmazione in linguaggio Basic. Visto pero' l'impegno eccessivamente oneroso, in termini di tempo e denaro, decisi che era meglio abusare della pazienza di un amico computerista, gia' possessore di un PET, con il risultato di imparare ugualmente, e senza spesa alcuna.
Infatti cosi' e' stato.
Passati i primi scogli dell'apprendimento del BASIC, mi sono accorto di quanto siano vaste le possibilita' di questi piccoli calcolatori che, andando ben oltre al puro divertimento, sono capaci di soddisfare perfettamente molte necessita' ad esempio sia di una casa o di un ufficio.
E' certo che entro pochi anni le comuni massaie ne faranno uso, cosi' come oggi usano il telecomando del televisore.
Allora, signori, vorrei dire: incominciamo a capirci un po' di piu', diamoci un po' piu' da fare per precorrere i tempi.Cio' che si prefigge la tecnologia del domani, e'la completa automazione in ogni attivita'. I rapidissimi progressi compiuti nell'elettronica ne sono la testimonianza. Non ci si metta in mente pero' l'idea, peraltro fantascientifica, che possa avvenire un sopravvento della mente "artificiale" su quella umana. Lasciamo alla fantasia questo genere di idee.
Ben venga quindi questo nuovo "personal computer", quale e' il "VIC 20". Tramite esso poi, ho scoperto un'eroica schiera di appassionati, che molto stanno facendo per diffondere l'informatica a livello di massa sia come hobby che come mezzo complementare al lavoro. Tutto cio' non fa altro che allargare il mio bagaglio di conoscenze e perche' no?..... di esperienze.
Un mondo nuovo,insomma, mi si presenta davanti, "aspettando il VIC ...."

VIC 20

Nella seguente tabella vengono riportate alcune delle piu' importanti locazioni di memoria del VIC20 con particolare puntualizzazione alla pagina zero ed alle corrispondenze con i PET CBM versione BASIC 4.0.
Si tenga presente che la versione BASIC 1.0 del VIC20 corrisponde al BASIC 4.0 dei PET CBM 4032 e 8032 con l'esclusione dei comandi tipo DLOAD DSAVE ed altri inerenti alla gestione dischi.

locazione decimale	descrizione locazione del V I C 2 0	corrisp. PET 4.0
0 2	locazioni per USR	0 2
43 44	puntatore inizio BASIC	40 41
45 46	puntatore fine BASIC e inizio variabili	42 43
47 48	puntatore fine variabili e inizio ARRAY	44 45
49 50	puntatore fine ARRAY	46 47
51 52	puntatore inizio zona stringhe	48 49
53 54	puntatore fine zona stringhe	50 51
55 56	puntatore fine memoria utente	52 53
57 58	numero dell'attuale linea BASIC	54 55
115 138	routine riporto nuovo carattere BASIC	112 135
144	byte di stato di I/O (ST)	150
197	immagine dell'ultimo tasto premuto	151
198	numero del carattere nel buffer di tastiera	158
199	flag di reverse (0=spento 18=acceso)	159
203	codice input della tastiera (64= nessun tasto)	166
204	flag di abilitaz. cursore (0=acceso 1=spento)	167
205	ritardo lampeggio cursore	168
206	carattere sotto il cursore	169
207	flag di cursore acceso	170
209 210	puntatore della linea sullo schermo	196 197
211	posizione del cursore sulla linea	198
214	riga su cui giace il cursore	161
245	flag tasti di controllo (159=shift 224=commod.	155
631 640	buffer di tastiera	623 632
650	funzione di repeat (128=ins. 127=dis. 0=parz.)	
651	ritardo che precede il repeat	
652	ritardo tra i repeat	
828 1019	buffer cassetta	634 825
1024 4095	3K RAM espansione	
4096 8191	normale area memoria RAM	
8192 32767	area di espansione RAM	
32768 36863	mappa dei caratteri	
36864 36879	VIC Video Interface Cip	
37136 37167	VIA 6522	
37888 38399	mappa video colore di schermo	
38400 38911	mappa colore	
40960 49151	area espansione ROM (plug-in)	
49152 65535	ROM BASIC e sistema operativo (versione 1.0)	

```
10 REM ******************************
20 REM *** OROLOGIO DEL VIC20   ***
30 REM ***     POCKET GROUP     ***
40 REM ***    STEFANO  MIARI    ***
50 REM ******************************
60
100 PRINT "[CLR]QQQQQQQQQQ"
110 FOR T=4 TO 13
120 POKE 7880+T,42
130 NEXT T
140 FOR U=4 TO 13
150 POKE 38600+U,0
160 NEXT U
170 FOR I=8 TO 17
180 POKE 7920+I,42
190 NEXT I
200 FOR Y=8 TO 17
210 POKE 38640+Y,0
220 NEXT Y
230 PRINT "[HOME]" TAB(7) LEFT$(TI$,2) ":";
240 PRINT MID$(TI$,3,2) ":";
250 PRINT RIGHT$(TI$,2)
260 P=VAL(RIGHT$(TI$,1))
270 C=INT(RND(1)*7)
280 IF C=1 THEN 270
290 IF A=P THEN 230
300 POKE 38692+P,C
310 POKE 7972+P,224
320 POKE 36879,24+C
330 POKE 36878,10
340 POKE 36876,220
350 A=P
360 POKE 36878,0
370 GOTO 230
```

VIC20

Il VIC20, come ogni altro computer che si rispetti, puo' svolgere numerose funzioni di cui spesso non si conosce l'esistenza.
Ecco allora per i possessori di questo "piccolo potente" un breve programmino che mostra come si puo' visualizzare sullo schermo del vostro televisore a colori (se possibile) il trascorrere del tempo.
Una delle variabili riservate al computer e' la TI$ che fornisce in qualsiasi momento il tempo in ore minuti e secondi.
Al momento dell'accensione del VIC20 si dovra' dire al nostro computer che ore sono; questo e' possibile impostando, in maniera diretta, TI$="223741", se in quel momento saranno le dieci e trentasette di sera; in sintesi si definisce l'orario in HHMMSS.
Dopo aver premuto il tasto di RETURN il VIC20 aggiornera' il proprio clock con il nuovo valore che verra' continuamente aggiornato automaticamente dal sistema.

Ma vediamo ora di capire cosa viene eseguito ad ogni riga di programma.

REMarks.

righe 10,20,30,40,50 contengono dei puri commenti (REM) che servono esclusivamente di presentazione, ma che non influsicono assolutamente con il programma se non per lunghezza.
riga 100 i segni particolari posti fra gli apici ci permettono di pulire lo schermo (tasto CLR con SHIFT) e di schendere con il cursore tante volte quante sono le lettere Q in reverse (tasto CRSR in giu').
righe 110,120,130 e 140,150,160 e similarmente 170,180,190 e 200,210,220 ci permettono di disegnare sullo schermo una pseudo cornice. I valori in 7880 identificano il carattere della cornice, mentre i valori posti in 38600 definiscono il colore del carattere scelto.
righe 230,240,250 si esegue l'edit dell'orario, separando opportunamente le ore dai minuti e i minuti dai secondi tramite il segno di duepunti.
riga 260 si assegna il valore alla variabile numerica P che corrispondera' al valore numerico dell'ultimo byte di TI$.
riga 270 assegnazione di un valore casuale compreso da zero a sette che ci permettera' di avere il codice del colore per la scansione visualizzata.
righe 280 fino alla fine; vengono disegnate sullo schermo, con i valori di P e C, le varie visualizzazioni colorate del passare del tempo e contemporaneamente viene eseguita anche l'emissione sonora.

Pensiamo di farvi piacere elencando le varie parti che compongono il sistema VIC.

VIC 20 Computer da 3.5k RAM utente, 24 colori in alta risoluzione, tastiera alfanumericografica, tasti di funzione, user port, porta seriale a 4800 Bauds, tre generazioni di suoni con estensione di tre ottave.
L. 590.000+IVA

VIC 1515 Stampante a matrice VIC DOT da 80 colonne, velocita' di stampa 40 carateri al secondo, alfanumericografica, seriale monodirezionale, allacciabile direttamente al VIC20.
Prossimo arrivo.

VIC 1530 Datacassette VIC per la lettura e memorizzazione dei dati, compatibile PET-CBM, con contatore di giri.
L. 120.000+IVA

VIC 1540 Unitá intelligente a singolo floppy disk da 5"1/4, della capacita' di 170.000 caratteri, a singola densita' e singola traccia, compatibile PET-CBM.
Prossimo arrivo.

VIC 1210 Cartuccia di espansione di memoria per VIC20 da 3k RAM.
Prossimo arrivo.

VIC 20

VIC 1110 Cartuccia di espansione di memoria per VIC20 da 8k RAM.

Prossimo arrivo.

VIC 20

VIC 1111 Cartuccia di espansione di memoria per VIC20 da 16k RAM.

Prossimo arrivo.

VIC 20

VIC 1212 Cartuccia di aiuto alla programmazione comprendente "Tool Kit" per programmatori, monitor di lunguaggio macchina, funzioni preassegnate per tasti, possibilita' di assegnare funzioni ai tasti direttamente dall'utente.

Prossimo arrivo.

VIC 20

VIC 1010 Cabinet di espansione per VIC20 per contenere fino ad un massimo di 6 schede di espansione contemporaneamente.

Prossimo arrivo.

VIC 20

VIC 1011a Interfaccia RS232 ad alta velocita' di trasmissione.

Prossimo arrivo.

VIC 20

VIC 1011b Come VIC 1011a in versione TTY

Prossimo arrivo.

VIC 20

VIC 1211 Super Expander Cartridge per alta risoluzione grafica (32.384 punti di video in totale), funzioni per tasti preassegnate, scrolling orizzontale.

Prossimo arrivo.

VIC 20

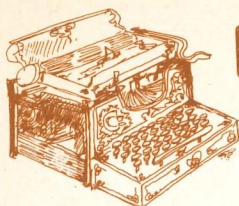
VIC 1211n Come VIC1211, ma con aggiunta di ampliamento di memoria 3k RAM.

Prossimo arrivo.

VIC 20

VIC 19xx Giochi vari su cartuccia ad alta risoluzione grafica e con effetti sonori.

Prossimo arrivo.

VIC 20

EDITOR MAX 2°

di Massimo Rossi

Siamo lieti di potervi annunciare di aver mantenuto la promessa fattavi sul numero scorso di POKET PET. Infatti siamo riusciti, e, vi assicuriamo, non e' stato facile, a implementare il programma EDITOR MAX, in modo che possa caricare le pagine desiderate, senza dover attendere che il registratore si fermi alla fine di ogni stringa.

Questo difetto, infatti, era una spina nel fianco, per chi scrive, in quanto rendeva laborioso e noioso, il salvataggio del testo, senza contare lo spreco di nastro che si faceva tra una stringa e l'altra. La soluzione di questo problema, era di caricare la parte di memoria in cui il programma viene immagazzinato, direttamente su nastro, come se fosse un programma, in un file di tipo ASCII, tale e quale a quello usato per mettere su cssetta i programmi in linguaggio macchina. Ed e' proprio usando le routine di monitor di linguaggio macchina, che abbiamo ottenuto il nostro scopo.

Abbiamo trovato, infatti, una routine di sistema operativo, che esegue la OPEN, per il SAVE, che e' in comune sia al monitor che al Basic, e che si trova all'indirizzo esadecimale $F6A4. Se si richiamasse questa routine dal Basic, ammesso che si possa, si avrebbe un salvataggio del testo basic, poiche' i puntatori di inizio e fine testo, sono automaticamente settati sulle dimensioni del testo Basic. Quello che bisogna fare, allora, e' di forzare i contenuti di queste locazioni (251-252, indirizzo di inizio testo; 201-202, indirizzo di fine testo), con gli indirizzi di inizio e fine del testo che vogliamo registrare. Questo si puo' solo fare con delle istruzioni in linguaggio macchina, alla fine delle quali, verra' richiamata la routine di OPEN.

Se noi volessimo usare delle POKE, non otterremmo nessun effetto, poiche'al richiamo della OPEN da Basic, queste locazioni verrebbero resettate dal sistema operativo, vanificando il nostro lavoro.

Ma prima di richiamare la OPEN, dovremo specificare altre cose: il numero della periferica, ad esempio. Questo, per la cassetta, e' "1", mentre, come vedremo poi, per il disco sarebbe "8". Forzeremo questo valore nella locazione 212, caricandolo nell'accumulatore e scaricandolo in questa locazione.

Un'altra necessaria informazione, e' il nome del file, in modo che lo si possa ritrovare anche su di un nastro dove esistono molti files in fila. La routine di OPEN, richiede che le si comunichi un indirizzo, in cui sia immagazzinato in codice ASCII, seguito da uno "00", il nome del file. Abbiamo percio' destinato uno spazio apposito, alla fine della routinetta che si trova nel buffer della seconda cassetta, di 10 Bytes, che conterra il nome del file, e la cui locazione di inizio sara' immagazzinata nell locazioni 218 e 219.

La routine, infine, ci chiede di specificare la lunghezza del nome del file, cosa che noi faremo, inserendo questo valore nella locazione 209.

A questo punto, abbiamo voluto inserire anche una routine di caricamento, sempre in linguaggio macchina, che ci permettera' di fare una LOAD, senza dover interrompere il programma. Qui bastera' specificare la device, la lunghezza del nome del file, e l'indirizzo di quest'ultimo. Seguira' la routine di LOAD, che si trova in $F322.

Ottenute queste due routinette, e legatole assieme, le abbiamo disassemblate in modo da ottenere degli statements di DATA, che una volta "pokkati", nelle giuste locazioni, da una subroutine inclusa nel programma, che viene richiamata ad ogni RUN, ricostruiscono le routines nel buffer della seconda cassetta, pronte per essere usate.

EDITOR MAX II LOADER PER CASSETTA

IND.N	ESA	DATI	MNEMON.	
826	033A	A9 01	LDAIM	1
828	033C	85 D4	STAZ	212
830	033E	A9 68	LDAIM	104
832	0340	85 DA	STAZ	218
834	0342	A9 03	LDAIM	3
836	0344	85 DB	STAZ	219
838	0346	A9 06	LDAIM	6
840	0348	85 D1	STAZ	209
842	034A	A9 0B	LDAIM	11
844	034C	85 FC	STAZ	252
846	034E	A9 B8	LDAIM	184
848	0350	85 FB	STAZ	251
850	0352	20 97 E7	JSR	59287
853	0355	A9 0F	LDAIM	15
855	0357	85 FC	STAZ	252
857	0359	85 CA	STAZ	202
859	035B	A9 50	LDAIM	80
861	035D	85 FB	STAZ	251
863	035F	85 C9	STAZ	201
865	0361	20 97 E7	JSR	59287
868	0364	20 A4 F6	JSR	63140
871	0367	60	RTS	
872	0368	45 44	EORZ	68
874	036A	49 54	EORIM	84
876	036C	4F*		
877	036D	52*		
878	036E	00	BRK	
879	036F	00	BRK	
880	0370	00	BRK	
881	0371	00	BRK	
882	0372	00	BRK	
883	0373	A9 01	LDAIM	1
885	0375	85 D4	STAZ	212
887	0377	A9 68	LDAIM	104
889	0379	85 DA	STAZ	218
891	037B	A9 03	LDAIM	3
893	037D	85 DB	STAZ	219
895	037F	A9 06	LDAIM	6
897	0381	85 D1	STAZ	209
899	0383	A2 00	LDXIM	0
901	0385	86 96	STX	150
903	0387	20 22 F3	JSR	62242
906	038A	60	RTS	

EDITOR MAX II LOADER PER DISCO

IND.N	ESA	DATI	MNEMON.	
826	033A	A9 08	LDAIM	8
828	033C	85 D4	STAZ	212
830	033E	A9 68	LDAIM	104
832	0340	85 DA	STAZ	218
834	0342	A9 03	LDAIM	3
836	0344	85 DB	STAZ	219
838	0346	A9 06	LDAIM	6
840	0348	85 D1	STAZ	209
842	034A	A9 0B	LDAIM	11
844	034C	85 FC	STAZ	252
846	034E	A9 B8	LDAIM	184
848	0350	85 FB	STAZ	251
850	0352	20 97 E7	JSR	59287
853	0355	A9 0F	LDAIM	15
855	0357	85 CA	STAZ	202
857	0359	85 CA	STAZ	202
859	035B	A9 50	LDAIM	80
861	035D	85 C9	STAZ	201
863	035F	85 C9	STAZ	201
865	0361	20 97 E7	JSR	59287
868	0364	20 A4 F6	JSR	63140
871	0367	60	RTS	
872	0368	30 3A	BMI	58
874	036A	00	BRK	
875	036B	00	BRK	
876	036C	00	BRK	
877	036D	52*		
878	036E	00	BRK	
879	036F	00	BRK	
880	0370	00	BRK	
881	0371	00	BRK	
882	0372	00	BRK	
883	0373	A9 08	LDAIM	8
885	0375	85 D4	STAZ	212
887	0377	A9 6A	LDAIM	106
889	0379	85 DA	STAZ	218
891	037B	A9 03	LDAIM	3
893	037D	85 DB	STAZ	219
895	037F	A9 06	LDAIM	6
897	0381	85 D1	STAZ	209
899	0383	A2 00	LDXIM	0
901	0385	86 96	STX	150
903	0387	20 22 F3	JSR	62242
906	038A	60	RTS	

Ovviamente, vi sono dei dati che cambiano ogni volta che si devono registrare le pagine. Per quanto riguarda il numero di pagine da registrare, cioe' l'indirizzo di fine registrazione, questo viene calcolato dal Basic, e "pokkato", una volta tradotto nei due valori, alto e basso, nelle locazioni relative.

Un'altra variabile, e' il nome del file: questo deve venir tradotto in ASCII, misurato nella lunghezza,(che viene caricata in 209), e a sua volta viene "pokkato" nelle celle libere apposite, fatto seguire da uno "00"

Un processo analogo, viene eseguito per il LOAD.

Una notazione interessante, e' che queste routines, non possono essere richiamate durante un programma, da una SYS, ma devono essere chiamate in "command mode", cioe' da comandi diretti.

Come fare allora, dato che non volevamo che il programma si interrompesse? La soluzione, anche se forse un po' macchinosa, e' l'unica che siamo

```
500 *GOSUB9000
1000 PM=INT((FRE(0)-1000)/920):P=0:M=0
1100 PRINT"■SONO DISPONIBILI"PM"PAGINE":FORI=1TO2000:NEXT
1200 POKE167,1:PRINT"■"
1300 POKE216,0:POKE198,0:SYS57949:PRINT"×";:FORI=1TO29:PRINT" ";:NEXT:PRINT
1400 IFM=0THEN1600
1500 POKE216,0:POKE198,32:SYS57949:PRINT"PAG."P                          "
1600 FORI=1TO40:POKE32807+I,99:NEXT:PRINT
1700 POKE167,0
1800 GETA$:IFA$=""THEN1800
1900 IFPEEK(170)=1THEN1900
2000 PRINTA$;:IFPEEK(216)>23THENPOKE216,23
2100 IFPEEK(216)<2ANDPEEK(151)=6THENGOTO1300
2200 IFPEEK(216)<2ANDPEEK(151)=65THENGOTO1300
2300 IFPEEK(32768)=0THEN2500
2400 GOTO1700
2500 POKE216,0:POKE198,12:SYS57949:PRINT"■ ATTENDO ORDINI■";
2600 POKE32768,32:POKE32769,32
2700 GETA$:IFA$=""THEN2700
2800 FORI=0TO39:POKE32768+I,32:NEXT
2900 IFA$="M"THEN3600
3000 IFA$="R"THEN4900
3100 IFA$="S"THEN6200
3200 IFA$="L"THEN6900
3300 IFA$="P"THENGOSUB7900:GOTO1200
3400 IFA$="C"THENM=0:GOTO1200
3500 GOTO2500
3600 POKE216,0:POKE198,5:SYS57949:PRINT"■MEMORIZZA■";
3700 POKE216,0:POKE198,21:SYS57949:PRINT"                    "
3800 POKE216,0:POKE198,21:SYS57949:INPUT"NUM.PAGINA";P
3900 POKE216,0:POKE198,21:SYS57949:PRINT"                    "
4000 IFP<PMTHEN4500
4100 POKE216,0:POKE198,21:SYS57949:PRINT"NON ESISTE"
4200 FORI=1TO1000:NEXT
4300 POKE216,0:POKE198,21:SYS57949:PRINT"                    "
4400 GOTO3800
4500 POKE216,0:POKE198,27:SYS57949:PRINT"MEM.PAG."P
4600 FORI=0TO919
4700 *POKE3600+(P*920)+I,PEEK(32848+I):NEXT
4800 M=0:GOTO1200
4900 POKE216,0:POKE198,5:SYS57949:PRINT"■RICHIAMA■";
5000 POKE216,0:POKE198,21:SYS57949:PRINT"                    "
5100 POKE216,0:POKE198,21:SYS57949:INPUT"NUM.PAGINA";P
5200 POKE216,0:POKE198,21:SYS57949:PRINT"                    "
5300 IFP<PMTHEN5700
5400 POKE216,0:POKE198,21:SYS57949:PRINT"NON ESISTE"
5500 FORI=1TO1000:NEXT
5600 POKE216,0:POKE198,21:SYS57949:PRINT"                    ":GOTO5100
5700 POKE216,0:POKE198,27:SYS57949:PRINT"RIC.PAG."P
5800 FORI=0TO919
5900 *POKE32848+I,PEEK(3600+(P*920)+I):NEXT
6000 POKE216,0:POKE198,28:SYS57949:PRINT"           "
6100 M=1:GOTO1300
6200 *GOSUB7700:GOSUB7100
6300 *D=3600+K*920:M=0
6400 *H%=D/256:R%=D-H%*256
6500 *POKE854,H%:POKE860,R%
6600 *PRINT"■■■■?CHR$(147):SYS826:GOTO1200■":POKE158,1:POKE623,13:END
6900 *GOSUB7800:GOSUB7100
7000 *PRINT"■■■■?CHR$(147):SYS883:GOTO1200■":POKE158,1:POKE623,13:END
7100 *LN=LEN(TS$):IFLN>10THENLN=10
7200 *POKE839,LN:POKE896,LN
7300 *FORI=1TOLN:F$=MID$(TS$,I,1):F=ASC(F$):POKE871+I,F:NEXT:POKE872+LN,0:RETURN
```

```
7700 *INPUT"[illegible]FINO A QUALE PAGINA";K
7800 *INPUT"[illegible]TITOLO TESTO";TS$:RETURN
7900 N=0:XT$="":OPEN4,4
8000 *FORI=32848TO33767
8100 GOSUB8400:XT$=XT$+X$
8200 N=N+1:IFN>39THENGOSUB8500:N=0
8300 *NEXT:CLOSE4:M=0:RETURN
8400 X=PEEK(I):X=(XAND127)OR((XAND64)*2)OR((64-XAND32)*2):X$=CHR$(X):RETURN
8500 *PRINT#4,XT$:XT$="":FR=FRE(0):RETURN
9000 *FORI=826TO906:READD:POKEI,D:NEXT:RETURN
9100 *DATA 169,1,133,212,169,104,133,218,169,3,133,219,169,6,133,209
9200 *DATA 169,11,133,252,169,184,133,251,32,151,231,169,15,133,252,133
9300 *DATA 202,169,80,133,251,133,201,32,151,231,32,164,246,96,69,68
9400 *DATA 73,84,79,82,0,0,0,0,0,169,1,133,212,169,104,133
9500 *DATA 218,169,3,133,219,169,6,133,209,162,0,134,150,32,34,243
9600 *DATA 96
```

CORREZIONI PER DISCO

```
7200 POKE839,LN+2:POKE896,LN
7300 FORI=1TOLN:F$=MID$(TS$,I,1):F=ASC(F$):POKE873+I,F:NEXT:POKE874+LN,0:RETURN
```

DATA DEL LOADER PER DISCO

```
9100 DATA 169,8,133,212,169,104,133,218,169,3,133,219,169,6,133,209
9200 DATA 169,11,133,252,169,184,133,251,32,151,231,169,15,133,202,133
9300 DATA 202,169,80,133,201,133,201,32,151,231,32,164,246,96,48,58
9400 DATA 0,0,0,0,0,0,0,0,0,169,8,133,212,169,106,133
9500 DATA 218,169,3,133,219,169,6,133,209,162,0,134,150,32,34,243
9600 DATA 96
```

riusciti a trovare. Facciamo scrivere sullo schermo lo statement da eseguire (nel nostro caso, la SYS, preceduta da un comando di cancellatura schermo CHR$(147), e seguita dal GOTO che riprende il programma), e lo facciamo seguire da due comandi, che forzano un RETURN. Questo viene fatto con un POKE 158,1, che dice quanti caratteri sono presenti nel buffer della tastiera, e con un POKE 623,13, dove 623 e' la prima posizione libera del buffer, e 13 il RETURN.

Questo fara' scrivere sullo schermo, lo statement da eseguire, e lo cancellera' subito, eseguendolo in command mode, non essendo preceduto da alcun numero.

Il testo del nuovo editor, e' molto simile a quello di quello vecchio, bastera' cancellare alcune righe, e sostituirne altre. Le righe da cancellare, sono:

6700,6800,7400,7500,7600

Le righe da sostituire, sono invece, quelle precedute da un asterisco (*), nel nuovo testo.

Alcune righe, sono state cambiate, in quanto alcuni lettori, ci avevano segnalato dei malfunzionamenti della routine di stampa. L'uso dell'opzione di stampa, infatti, cancellava tutte le pagine scritte precedentemente. Questo era dovuto al fatto che la riga 8100, generava una stringa di nome XT$, che poi veniva sostituita da un'altra stringa con lo stesso nome: questo non fa si', come avviene per le altre variabili, che la nuova stringa vada a sostituire la precedente nellea memoria, ma bensi', l'ultima sposta in giu' la precedente, occupando progressivamente lo spazio delle nostre pagine. L'unico modo per evitare cio', e' di forzare la FRE(0), che e' una routine che ripulisce la memoria dalle vecchie stringhe, e che di solito,

viene richiamata solo quando la memoria e' piena. Noi la richiamiamo, uguagliandola a una variabile fittizia FR. Questo forza la routine di Garbage Collection, ad ogni fine stringa, evitandoci i fastidi di cui sopra.
Diamo un'occhiata al programma Basic:

500	Rimanda alla subroutine di loader in 9000
4700 5900	Abbiamo dovuto partire piu' in alto con la memorizzazione, poiche' il programma e' piu' lungo
6200-6500	Computa l'indirizzo di arrivo
6600	Richiama la routine di SAVE
7000	Richiama la routine di LOAD
7100-7300	Sistema il nome del file e la sua lunghezza
7700-7800	Inputs varii
8000	Inizio stampa alla riga corretta
8300	Corregge la variabile M
8500	Richiama il Garbage Collection, dopo ogni riga stmpata
9000-9600	Routine di loader

Diamo, ora, un'occhiata alla routine in linguaggio macchina.
Abbiamo usato dei comandi di load diretto (LDA), nell'accumulatore, per poi immagazzinare i dati contenuti in quest'ultimo, nelle locazioni volute, con uno STA, cioe' "metti il contenuto dell'accumulatore nella locazione voluta". Abbiamo, poi, immesso i richiami alle routines di caricamento su nastro, che sono di seguito $E797 e $F6A4. Terminata l'esecuzione del programma di save, l'istruzione 60, (RTS), riporta il controllo al Basic. Troviamo poi, in locazione dec. 872-882, lo spazio per il salvataggio del nome del file (nella routine e' scritto EDITIR, ma durante l'uso, si immettera' il nome desiderato).
Vi e', quindi, la routine di load, costruita come la precedente, con il richiamo alla subroutine di load che si trova in locazione esad. $F322.
Termina qui la descrizione di questa implementazione, che, cronometro alla mano, ci risulta impiegare per il caricamento di ogni pagina, circa 15 secondi, contro i 2 minuti circa, della versione precedente. Un buon risultato, non vi sembra?
Ma non ci siamo accontentati di questo: ora vi presentiamo una versione del nuovo EDITOR MAX, per chi ha il Floppy Disk Driver, e desidera un caricamento rapido, e che occupa poco spazio sul disco.
Dovete solo correggere le righe 7200 e 7300, secondo la figura fuori testo, e sostituire alle righe di DATA, nel testo del loader per cassetta, le righe in figura.
In locazione 872, vi e' 30, che in codice ASCII, e'"0". Questo e' il numero del drive. Se qualche lettore lo desidera, puo' inserire nel testo Basic, un input per scegliere il drive: si pokkera', poi in locazione 872, 30, se il driver e' quello di destra, e 31, se il driver e' quello di sinistra.
Tutto il resto del funzionamento, e' identico al primo EDITOR MAX, che fu pubblicato nel numero 2 di Pocket Pet.
Speriamo di avervi fatto piacere, e vi chiediamo, comunque di inviarci Vostri eventuali suggerimenti, che saranno sempre ben accolti, e premiati. Nuove implementazioni, sono in progettazione.

N.B.: Per chi volesse adattare l'EDITOR MAX per le versioni BASIC diverse dal 3.0, cioe' per le vecchie ROM e per il BASIC 4.0 suggeriamo di consultare le note riportate nel presente numero nell'articolo: "A proposito del RAGNO NERO".
Ricordiamo anche che per una svista del proto, nella scorsa puntata, il carattere "@" e' stato sostituito erroneamente con il carattere "%".

Nuovi comandi per il BASIC

di Roberto Sozzani

Questo articolo e' tratto da un articolo di David Simons apparso su un numero di quest'anno di una rivista inglese.

In esso si spiegava come si possono aggiungere dei comandi al BASIC, ed era completato da un programma dimostrativo. Sulla scorta di quelle note ho elaborato un programma per aggiungere una serie di nuovi comandi al BASIC. Esso puo' servire da esempio sulla procedura necessaria, ed ognuno e' libero di aggiungere le funzioni che ritiene piu' utili o di cancellare quelle di poco interesse. Ogni comando, in pratica, rimanda ad una routine in linguaggio macchina residente nella parte alta della memoria. Si possono pertanto costruire programmi Basic in cui intere funzioni vengono svolte da routine in linguaggio macchina richiamate da semplici comandi. Cio' semplifica notevolmente il compito del programmatore e rende il listato incomprensibile al profano. Vediamo ora come si possono fare accettare al PET dei nuovi comandi. Bisogna innanzitutto sapere che alla locazione $70 inizia una routine, chiamata CHARGET. Essa analizza le linee BASIC, stabilendo se accettarle, o rifiutarle con il fatidico messaggio "SYNTAX ERROR". Ecco come si presenta questa routine:

```
0070 E6 77     INC $0077 ; incrementa byte sorgente
0072 D0 02     BNE $0076 ; nuova pagina ?
0074 E6 78     INC $0078 ; si, incrementa il contatore di pagina
0076 AD 02 02  LDA $0202 ; carica il contenuto della
                           locazione $0202
0079 C9 3A     CMP #$3A  ; e' un due punti ?
007B B0 0A     BCS $0087 ; se il carry e' settato, va a RETURN
007D C9 20     CMP #$20  ; e' uno spazio ?
007F F0 EF     BEQ $0070 ; si, ricomincia
0081 38        SEC       ; setta il carry
0082 E9 30     SBC #$30  ; sottrae il numero DEC 48
0084 38        SEC       ; setta il carry
0085 E9 D0     SBC #$D0  ; sottrae il numero DEC 208
0087 60        RTS       ; RETURN
```

Come si vede, essa controlla i vari caratteri della parola Basic, e poi prosegue per stabilire se il carattere successivo sia un due punti o

uno spazio; questo pero' per il momento non ci interessa. Interessa invece sapere che noi possiamo modificare il contenuto della CHARGET in modo da farla rimandare alla locazione di memoria in cui risiede la nostra routine con i nuovi comandi. Se, ad esempio, questa locazione fosse $7000, la CHARGET diventera':

```
0070 4C 00 70  JMP $7000 ;goto $7000
```

Quando il PET va alla locazione $70 trova cosi' un'istruzione di salto alla locazione da noi scelta, ed invece di eseguire le normali operazioni di controllo dell'istruzione BASIC, eseguira' le nostre istruzioni. La routine per caricare quest'istruzione di salto e' molto semplice e puo' essere locata nella zona di memoria che piu' ci aggrada, come nel buffer della prima o della seconda cassetta, o direttamente subito prima del programma vero e proprio.

Nel nostro esempio abbiamo scelto questa ultima possibilita', e la routine di abilitazione, che inizia alla locazione $7800, e' esposta nelle righe 0019-0024 della pagina 1 del listato assembler, ed e' seguita da una breve routine di protezione della memoria in cui risiedera' il nostro programma (righe 0025-0031). Queste istruzioni pongono il valore $77FF nei puntatori di fine memoria RAM. Il programma verra' quindi abilitato con il comando SYS 30720 (che equivale a HEX 7800).

Le istruzioni delle righe 0035-0042 disabilitano il programma rilocando in memoria le originarie istruzioni della CHARGET.

Il programma vero e proprio inizia alla riga 0046, con un test per verificare se il carattere da noi digitato e' un nuovo comando. Quest'ultima operazione viene eseguita alla riga 0068 (CMP #$40). Il valore HEX 40 corrisponde al carattere "@". Tutti i nostri nuovi comandi dovranno pertanto iniziare con questo carattere. Se si volesse cambiare carattere, e' sufficiente porre dopo C9 il valore ASCII espresso in esadecimale del carattere scelto.

Dopo aver verificato che il carattere corrisponde ad un nuovo comando, il programma carica un nuovo byte per vedere di quale comando si tratti. Se trova un carattere previsto da programma esegue le relative istruzioni, altrimenti ritorna alla CHARGET e visualizza SYNTAX ERROR.

Maggiori dettagli sono contenuti nel listato. Il programma viene presentato con i valori relativi al Basic 4.0. I valori per il Basic 3.0 (nuove ROM) sono specificati, fra parentesi, nelle righe che necessitano la correzione. Ci tengo a ripetere che le varie funzioni aggiunte con questo programma hanno un carattere dimostrativo ed esplicativo e come tali vanno considerate. Spero serviranno di esempio per tutti coloro che vogliono cominciare a cimentarsi con il linguaggio macchina, e se qualcuno trovasse nuove funzioni particolarmente interessanti da aggiungere puo' spedirle alle redazione di questa rivista, che le pubblichera'.

COMANDI:

@I = Esegue il reverse dello schermo.
@# = Esegue la conversione da decimale a esadecimale.
Deve essere seguito dal numero da convertire.
@& = Abilita il comando di "repeat": un carattere viene ripetuto fino a quando il tasto e' premuto.
@S = Setta il suono: deve essere seguito dai valori, separati da una virgola, dell'ottava e della nota.
Es.: @S 15,250
@K = Disabilita il repeat ed azzera il suono.
@2 = Esegue lo scroll verso il basso dello schermo.
@8 = Esegue lo scroll verso l'alto dello schermo.
@Q = Disabilita il programma.
@A = Setta lo schermo sui caratteri maiuscoli.
@B = Setta lo schermo sui caratteri minuscoli.
@M = Mette in memoria l'intero schermo. I dati vengono memorizzati a partire dalla locazione HEX 7C00.
@L = Carica i dati dalla memoria e li visualizza sullo schermo.
@X = Scambia lo schermo con la memoria, ovvero i dati sullo schermo passano in memoria, ed i dati in memoria tornano sullo schermo.
@P = Stampa i mille caratteri presenti sullo schermo.
Deve essere saguito da:
1 = stampa allargata - 2 = stampa normale.
@C = Disegna una cornice sullo schermo: deve essere seguito dal valore di PEEK del carattere voluto.
Es.: @C 102

N.B.: Tutti i comandi che sono seguiti da dei numeri devono aver interposto uno spazio, altrimenti il comando non viene accettato.
Es.: @P1 = SYNTAX ERROR

@P 1 = Stampa allargata.

Come caricare l'EXTRA BASIC ?

Proprio nell'ultima pagina di questo articolo viene riportato sia un programmino di LOADER che la lista in linguaggio macchina dell'EXTRA BASIC.

Intuitivamente due sono i metodi di caricamento.

Il primo e' quello di riportare nei DATA tutti i valori esadecimali della lista in LM ricordandosi di porre come ultimo dato il valore -1; sara' il programmino stesso che provvedera' a tradurre ogni valore esadecimale in decimale e quindi a 'pokkarlo' nella giusta locazione. Dopo un po' di tempo, abbastanza lungo in quanto molti sono i DATA da pokkare, e' il medesimo programma che suggerisce la giusta SYS per attivare l'EXTRA BASIC.

Il secondo metodo e' quello di inserire direttamente i valori in LM tramite l'uso del monitor TIM:

```
RIGA# LOC   CODICE        IST.

0001  0000                ;*********************************
0002  0000                ;* EXTRA BASIC 4.0 - OTTOBRE 81 *
0003  0000                ;* R.S. - POCKET GROUP - MILANO *
0004  0000                ;* ( BASIC 3.0 TRA PARENTESI )  *
0005  0000                ;*********************************
0006  0000                ;
0007  0000                ;
0008  0000                FINE=$76
3009  0000                SUONO=59467
0010  0000                OTTAVA=59466
0011  0000                NOTA=59464
0012  0000                SCR=59468
0013  0000                START=30720
0014  0000                ;
0015  0000                        *=START
0016  7800                ;
0017  7800                ;ABILITAZIONE PROGRAMMA
0018  7800                ;
0019  7800  A9 4C                 LDA #$4C
0020  7802  85 70                 STA $70
0021  7804  A9 2D                 LDA #<PREP
0022  7806  85 71                 STA $71
0023  7808  A9 78                 LDA #>PREP
0024  780A  85 72                 STA $72
0025  780C  A9 77                 LDA #$77        ; PROTEGGE MEMORIA
0026  780E  85 35                 STA $35
0027  7810  85 31                 STA $31
0028  7812  A9 FF                 LDA #$FF
0029  7814  85 30                 STA $30
0030  7816  85 34                 STA $34
0031  7818  4C FF B3              JMP $B3FF       ; (4C 89 C3)  JUMP A READY
0032  781B                ;
0033  781B                ;DISABILITAZIONE PROGRAMMA
0034  781B                ;
0035  781B  A9 E6         DIS     LDA #$E6
0036  781D  85 70                 STA $70
0037  781F  A9 77                 LDA #$77
0038  7821  85 71                 STA $71
0039  7823  A9 D0                 LDA #$D0
0040  7825  85 72                 STA $72
0041  7827  20 E3 78              JSR SUB
0042  782A  20 FF B3              JSR $B3FF       ; (4C 89 C3)  JUMP A READY
0043  782D                ;
0044  782D                ;PREPARAZIONE
0045  782D                ;
0046  782D  8E 90 03      PREP    STX $0390       ; MEMORIZZA X
0047  7830  BA                    TSX
0048  7831  BD 01 01              LDA $0101,X     ; E' IN EDIT ?
0049  7834  C9 23                 CMP #$23        ; (C9 F9)
0050  7836  D0 10                 BNE OUT
0051  7838  BD 02 01              LDA $0102,X
0052  783B  C5 C6                 CMP $C6
0053  783D  F0 09                 BEQ OUT
0054  783F  E6 77                 INC $77         ; SI, INCREMENTA BYTE SORGENTE
0055  7841  D0 02                 BNE NO
```

782D INIZIO

```
RIGA# LOC   CODICE       IST.

0056  7843  E6 78                 INC $78
0057  7845  4C 54 78     NO       JMP TEST       ; VA A VEDERE SE NUOVO COMANDO
0058  7848  AE 90 03     OUT      LDX $0390      ; NO, RECUPERA X
0059  784B  E6 77                 INC $77        ; INCREMENTA BYTE SORGENTE
0060  784D  D0 02                 BNE NO1
0061  784F  E6 78                 INC $78
0062  7851  4C 76 00     NO1      JMP FINE       ; GET PROSSIMO BYTE
0063  7854  A0 00        TEST     LDY #00
0064  7856  B1 77                 LDA ($77),Y    ; GET CARATTERE
0065  7858               ;
0066  7858               ;E' UN NUOVO COMANDO ?
0067  7858               ;
0068  7858  C9 40                 CMP #$40       ; = @ ?
0069  785A  F0 03                 BEQ SI         ; SE SI PROSEGUE
0070  785C  4C 76 00              JMP FINE       ; ALTRIMENTI RICOMINCIA
0071  785F  E6 77        SI       INC $77
0072  7861  D0 02                 BNE NO2
0073  7863  E6 78                 INC $78
0074  7865  B1 77        NO2      LDA ($77),Y
0075  7867  E6 77                 INC $77
0076  7869  D0 02                 BNE NO3
0077  786B  E6 78                 INC $78
0078  786D               ;
0079  786D               ;REVERSE DELLO SCHERMO
0080  786D               ;
0081  786D  C9 49        NO3      CMP #$49       ; = I ?
0082  786F  D0 17                 BNE VIA
0083  7871  A2 80                 LDX #$80
0084  7873  A0 00                 LDY #$00
0085  7875  84 01                 STY $01
0086  7877  86 02        LP       STX $02
0087  7879  B1 01        LP1      LDA ($01),Y
0088  787B  49 80                 EOR #$80       ; REVERSE SINGOLO CARATTERE
0089  787D  91 01                 STA ($01),Y
0090  787F  C8                    INY
0091  7880  D0 F7                 BNE LP1
0092  7882  E8                    INX
0093  7883  E0 84                 CPX #$84
0094  7885  D0 F0                 BNE LP
0095  7887  60                    RTS
0096  7888               ;
0097  7888               ;CONVERTE DA DEC A HEX
0098  7888               ;
0099  7888  C9 23        VIA      CMP #$23       ; = # ?
0100  788A  D0 13                 BNE VIA1
0101  788C  20 84 BD              JSR $BD84      ; (20 8B CC)
0102  788F  20 2D C9              JSR $C92D      ; (20 D2 D6)
0103  7892  20 22 D7              JSR $D722      ; (20 75 E7)
0104  7895  98                    TYA
0105  7896  20 22 D7              JSR $D722      ; (20 75 E7)
0106  7899  D0 01                 BNE NO4
0107  789B  60                    RTS
0108  789C  4C 76 00     NO4      JMP FINE
0109  789F               ;
0110  789F               ;ABILITA REPEAT
```

```
RIGA# LOC   CODICE      IST.

0111  789F              ;
0112  789F  C9 26       VIA1   CMP #$26       ; = & ?
0113  78A1  D0 3C              BNE VIA2
0114  78A3  78                 SEI
0115  78A4  A9 B2              LDA #<REPEAT   ; MODIFICA IRQ
0116  78A6  85 90              STA $90
0117  78A8  A9 78              LDA #>REPEAT
0118  78AA  85 91              STA $91
0119  78AC  A9 01              LDA #1
0120  78AE  85 FF              STA $FF
0121  78B0  58                 CLI
0122  78B1  60                 RTS
0123  78B2  A5 97       REPEAT LDA $97
0124  78B4  C5 FD              CMP $FD
0125  78B6  F0 09              BEQ REP1
0126  78B8  85 FD              STA $FD
0127  78BA  A9 10              LDA #$10
0128  78BC  85 FE              STA $FE
0129  78BE  4C 55 E4    STOP   JMP $E455      ; (4C 2E E6)
0130  78C1  C9 FF       REP1   CMP #$FF
0131  78C3  F0 F9              BEQ STOP
0132  78C5  A5 FE              LDA $FE
0133  78C7  F0 04              BEQ REP2
0134  78C9  C6 FE              DEC $FE
0135  78CB  D0 F1              BNE STOP
0136  78CD  C6 FF       REP2   DEC $FF
0137  78CF  D0 ED              BNE STOP
0138  78D1  A9 02              LDA #2
0139  78D3  85 FF              STA $FF
0140  78D5  A9 00              LDA #0
0141  78D7  85 97              STA $97
0142  78D9  A9 02              LDA #2
0143  78DB  85 A8              STA $A8
0144  78DD  D0 DF              BNE STOP
0145  78DF              ;
0146  78DF              ;DISABILITA SUONO E REPEAT
0147  78DF              ;
0148  78DF  C9 4B       VIA2   CMP #$4B       ; = K ?
0149  78E1  D0 1A              BNE VIA3
0150  78E3  A9 55       SUB    LDA #$55       ; (A9 2E)
0151  78E5  85 90              STA $90
0152  78E7  A9 E4              LDA #$E4       ; (A9 E6)
0153  78E9  85 91              STA $91
0154  78EB  A9 10              LDA #$10
0155  78ED  85 FD              STA $FD
0156  78EF  A9 00              LDA #0
0157  78F1  8D 4B E8           STA SUONO
0158  78F4  8D 4A E8           STA OTTAVA
0159  78F7  8D 48 E8           STA NOTA
0160  78FA  85 01              STA $01
0161  78FC  60                 RTS
0162  78FD              ;
0163  78FD              ;ABILITA SUONO
0164  78FD              ;
0165  78FD  C9 53       VIA3   CMP #$53       ; = S ?
```

```
RIGA# LOC   CODICE      IST.

0166  78FF  D0 18                BNE VIA4
0167  7901  A9 10                LDA #16
0168  7903  8D 4B E8             STA SUONO
0169  7906  20 D1 C8             JSR $C8D1        ; (20 75 D6) GET BYTE
0170  7909  E4 01                CPX $01
0171  790B  F0 05                BEQ SI1
0172  790D  86 01                STX $01
0173  790F  8E 4A E8             STX OTTAVA
0174  7912  20 27 C9      SI1    JSR $C927        ; (20 CC D6) GET PARAMETRO
0175  7915  8E 48 E8             STX NOTA
0176  7918  60                   RTS
0177  7919                ;
0178  7919                ;SCROLL VERSO IL BASSO
0179  7919                ;
0180  7919  C9 32         VIA4   CMP #$32         ; = 2 ?
0181  791B  D0 2A                BNE VIA5
0182  791D  A0 28                LDY #$28
0183  791F  84 22                STY $22
0184  7921  A0 00                LDY #0
0185  7923  84 20                STY $20
0186  7925  A0 BF                LDY #$BF
0187  7927  A2 83                LDX #$83
0188  7929  86 21         LP2    STX $21
0189  792B  86 23                STX $23
0190  792D  B1 20         LP3    LDA ($20),Y
0191  792F  91 22                STA ($22),Y
0192  7931  88                   DEY
0193  7932  C0 FF                CPY #$FF
0194  7934  D0 F7                BNE LP3
0195  7936  CA                   DEX
0196  7937  E0 7F                CPX #$7F
0197  7939  D0 EE                BNE LP2
0198  793B  A0 27                LDY #$27
0199  793D  A9 20                LDA #$20
0200  793F  91 20         LP4    STA ($20),Y
0201  7941  88                   DEY
0202  7942  10 FB                BPL LP4
0203  7944  4C 76 00             JMP FINE
0204  7947                ;
0205  7947                ;SCROLL VERSO L'ALTO
0206  7947                ;
0207  7947  C9 38         VIA5   CMP #$38         ; = 8 ?
0208  7949  D0 04                BNE VIA6
0209  794B  20 69 E3             JSR $E369        ; (20 3F E5) ESEGUE SCROLL
0210  794E  60                   RTS
0211  794F                ;
0212  794F                ;DISABILITA IL PROGRAMMA
0213  794F                ;
0214  794F  C9 51         VIA6   CMP #$51         ; = Q ?
0215  7951  D0 03                BNE VIA7
0216  7953  4C 1B 78             JMP DIS
0217  7956                ;
0218  7956                ;UPPER CASE
0219  7956                ;
0220  7956  C9 41         VIA7   CMP #$41         ; = A ?
```

```
RIGA# LOC   CODICE       IST.

0221  7958  D0 05                 BNE VIA8
0222  795A  A9 0C                 LDA #12
0223  795C  8D 4C E8              STA SCR
0224  795F               ;
0225  795F               ;LOWER CASE
0226  795F               ;
0227  795F  C9 42        VIA8     CMP #$42        ; = B ?
0228  7961  D0 05                 BNE VIA9
0229  7963  A9 0E                 LDA #14
0230  7965  8D 4C E8              STA SCR
0231  7968               ;
0232  7968               ;MEMORIZZA SCHERMO
0233  7968               ;
0234  7968  C9 4D        VIA9     CMP #$4D        ; = M ?
0235  796A  D0 20                 BNE VIA10
0236  796C  18                    CLC
0237  796D  A2 80                 LDX #128
0238  796F  A0 00                 LDY #0
0239  7971  84 21                 STY $21
0240  7973  86 22        NO5      STX $22
0241  7975  A5 22                 LDA $22
0242  7977  E9 03                 SBC #3
0243  7979  85 01                 STA $01
0244  797B  A9 00                 LDA #0
0245  797D  85 00                 STA $00
0246  797F  B1 21        NO6      LDA ($21),Y
0247  7981  91 00                 STA ($00),Y
0248  7983  C8                    INY
0249  7984  D0 F9                 BNE NO6
0250  7986  E8                    INX
0251  7987  E0 84                 CPX #$84
0252  7989  D0 E8                 BNE NO5
0253  798B  60                    RTS
0254  798C               ;
0255  798C               ;CARICA DA MEMORIA
0256  798C               ;
0257  798C  C9 4C        VIA10    CMP #$4C        ; = L ?
0258  798E  D0 20                 BNE VIA11
0259  7990  18                    CLC
0260  7991  A2 80                 LDX #128
0261  7993  A0 00                 LDY #0
0262  7995  84 21                 STY $21
0263  7997  86 22        NO7      STX $22
0264  7999  A5 22                 LDA $22
0265  799B  E9 03                 SBC #3
0266  799D  85 01                 STA $01
0267  799F  A9 00                 LDA #0
0268  79A1  85 00                 STA $00
0269  79A3  B1 00        NO8      LDA ($00),Y
0270  79A5  91 21                 STA ($21),Y
0271  79A7  C8                    INY
0272  79A8  D0 F9                 BNE NO8
0273  79AA  E8                    INX
0274  79AB  E0 84                 CPX #$84
0275  79AD  D0 E8                 BNE NO7
```

```
RIGA# LOC   CODICE      IST.


0276  79AF  60                 RTS
0277  79B0              ;
0278  79B0              ;SCAMBIA SCHERMO/MEMORIA
0279  79B0              ;
0280  79B0  C9 58       VIA11  CMP #$58        ; = X ?
0281  79B2  D0 28              BNE VIA12
0282  79B4  18                 CLC
0283  79B5  A2 80              LDX #128
0284  79B7  A0 00              LDY #0
0285  79B9  84 21              STY $21
0286  79BB  86 22       NO9    STX $22
0287  79BD  A5 22              LDA $22
0288  79BF  E9 03              SBC #3
0289  79C1  85 01              STA $01
0290  79C3  A9 00              LDA #0
0291  79C5  85 00              STA $00
0292  79C7  B1 21       NO10   LDA ($21),Y
0293  79C9  85 02              STA $02
0294  79CB  B1 00              LDA ($00),Y
0295  79CD  91 21              STA ($21),Y
0296  79CF  A5 02              LDA $02
0297  79D1  91 00              STA ($00),Y
0298  79D3  C8                 INY
0299  79D4  D0 F1              BNE NO10
0300  79D6  E8                 INX
0301  79D7  E0 84              CPX #$84
0302  79D9  D0 E0              BNE NO9
0303  79DB  60                 RTS
0304  79DC              ;
0305  79DC              ;STAMPA SCHERMO
0306  79DC              ;
0307  79DC  C9 50       VIA12  CMP #$50        ; = P ?
0308  79DE  F0 03              BEQ PRINT
0309  79E0  4C 79 7A           JMP VIA13
0310  79E3  20 D1 C8    PRINT  JSR $C8D1       ; (20 75 D6) GET BYTE
0311  79E6  E0 03              CPX #3          ; E' MINORE DI 3 ?
0312  79E8  30 03              BMI MIN
0313  79EA  4C 76 00           JMP FINE
0314  79ED  8E 7A 02    MIN    STX $027A
0315  79F0  A9 04              LDA #4
0316  79F2  85 D2              STA $D2
0317  79F4  A9 FF              LDA #$FF        ; (A9 6F)
0318  79F6  85 D3              STA $D3
0319  79F8  A9 04              LDA #4
0320  79FA  85 D4              STA $D4
0321  79FC  A2 04              LDX #4
0322  79FE  20 63 F5           JSR $F563       ; (20 24 F5) OPEN
0323  7A01  A2 04              LDX #4
0324  7A03  20 C9 FF           JSR $FFC9       ; PREDISPONE DEVICE #4
0325  7A06  20 0D 7A           JSR PRINT1
0326  7A09  20 E7 FF           JSR $FFE7       ; CLOSE FILES
0327  7A0C  60                 RTS
0328  7A0D  A9 80       PRINT1 LDA #128
0329  7A0F  A0 00              LDY #0
0330  7A11  84 21              STY $21
```

```
RIGA# LOC   CODICE       IST.

0331  7A13  85 22              STA $22
0332  7A15  A2 00      RIC     LDX #0
0333  7A17  B1 21      NO11    LDA ($21),Y
0334  7A19  20 72 7A           JSR PRINT2
0335  7A1C  C9 40              CMP #$40           ; PREDISPOSIZIONE CARATTERE
0336  7A1E  30 24              BMI MI1
0337  7A20  C9 7E              CMP #$7E
0338  7A22  30 25              BMI MI2
0339  7A24  C9 80              CMP #128
0340  7A26  30 1C              BMI MI1
0341  7A28  C9 BF              CMP #$BF
0342  7A2A  30 22              BMI MI3
0343  7A2C  C9 FF              CMP #255
0344  7A2E  F0 23              BEQ SI2
0345  7A30  9D 7B 02   BUF     STA $027B,X
0346  7A33  E8                 INX
0347  7A34  E0 28              CPX #$28
0348  7A36  F0 20              BEQ SI3
0349  7A38  C8                 INY
0350  7A39  D0 DC              BNE NO11
0351  7A3B  E6 22              INC $22
0352  7A3D  A5 22              LDA $22
0353  7A3F  C9 84              CMP #$84
0354  7A41  D0 D4              BNE NO11
0355  7A43  60                 RTS
0356  7A44  69 40      MI1     ADC #64
0357  7A46  4C 30 7A           JMP BUF
0358  7A49  69 80      MI2     ADC #128
0359  7A4B  4C 30 7A           JMP BUF
0360  7A4E  E9 40      MI3     SBC #64
0361  7A50  4C 30 7A           JMP BUF
0362  7A53  A9 BF      SI2     LDA #$BF
0363  7A55  4C 30 7A           JMP BUF
0364  7A58  A9 0D      SI3     LDA #13
0365  7A5A  9D 7B 02           STA $027B,X
0366  7A5D  E8                 INX
0367  7A5E  A9 00              LDA #0
0368  7A60  9D 7B 02           STA $027B,X
0369  7A63  84 00              STY $00
0370  7A65  A9 7A              LDA #$7A
0371  7A67  A0 02              LDY #2
0372  7A69  20 1D BB           JSR $BB1D          ; (20 1C CA)-PRINT STRINGA
0373  7A6C  A4 00              LDY $00
0374  7A6E  C8                 INY
0375  7A6F  4C 15 7A           JMP RIC
0376  7A72  C9 80      PRINT2  CMP #128
0377  7A74  30 02              BMI MI
0378  7A76  E9 80              SBC #128
0379  7A78  60         MI      RTS
0380  7A79                     ;
0381  7A79                     ;DISEGNA CORNICE
0382  7A79                     ;
0383  7A79  C9 43      VIA13   CMP #$43           ; = C ?
0384  7A7B  F0 03              BEQ COR
0385  7A7D  4C 76 00           JMP FINE
```

```
RIGA# LOC   CODICE      IST.

0386  7A80  20 D1 C8   COR    JSR $C8D1       ; (20 75 D6) GET BYTE
0387  7A83  86 21             STX $21         ; E LO MEMORIZZA
0388  7A85  A9 00             LDA #0
0389  7A87  85 00             STA $00
0390  7A89  A9 80             LDA #$80
0391  7A8B  85 01             STA $01
0392  7A8D  A2 17             LDX #23
0393  7A8F  A0 00      LP8    LDY #0
0394  7A91  A5 21             LDA $21
0395  7A93  91 00      LP9    STA ($0),Y
0396  7A95  C8                INY
0397  7A96  C0 28             CPY #40
0398  7A98  D0 F9             BNE LP9
0399  7A9A  E0 17             CPX #23
0400  7A9C  D0 21             BNE OUT1
0401  7A9E  A0 00      LP10   LDY #0
0402  7AA0  18                CLC
0403  7AA1  A5 00             LDA $0
0404  7AA3  69 28             ADC #40
0405  7AA5  85 00             STA $0
0406  7AA7  90 02             BCC CLR
0407  7AA9  E6 01             INC $01
0408  7AAB  A5 21      CLR    LDA $21
0409  7AAD  91 00             STA ($0),Y
0410  7AAF  A0 27             LDY #39
0411  7AB1  91 00             STA ($0),Y
0412  7AB3  CA                DEX
0413  7AB4  D0 E8             BNE LP10
0414  7AB6  A5 00             LDA $0
0415  7AB8  69 28             ADC #40
0416  7ABA  85 00             STA $0
0417  7ABC  4C 8F 7A          JMP LP8
0418  7ABF  4C 76 00   OUT1   JMP FINE
0419  7AC2                    .END

ERRORI = 0000

TAB. SIMBOLI

VALORE SIMB.
 BUF     7A30   CLR     7AAB   COR     7A80   DIS     781B
 FINE    0076   LP      7877   LP1     7879   LP10    7A9E
 LP2     7929   LP3     792D   LP4     793F   LP8     7A8F
 LP9     7A93   MI      7A78   MI1     7A44   MI2     7A49
 MI3     7A4E   MIN     79ED   NO      7845   NO1     7851
 NO10    79C7   NO11    7A17   NO2     7865   NO3     786D
 NO4     789C   NO5     7973   NO6     797F   NO7     7997
 NO8     79A3   NO9     79BB   NOTA    E848   OTTAVA  E84A
 OUT     7848   OUT1    7ABF   PREP    782D   PRINT   79E3
 PRINT1  7A0D   PRINT2  7A72   REP1    78C1   REP2    78CD
 REPEAT  78B2   RIC     7A15   SCR     E84C   SI      785F
 SI1     7912   SI2     7A53   SI3     7A58   START   7800
 STOP    78BE   SUB     78E3   SUONO   E84B   TEST    7854
 VIA     7888   VIA1    789F   VIA10   798C   VIA11   79B0
 VIA12   79DC   VIA13   7A79   VIA2    78DF   VIA3    78FD
 VIA4    7919   VIA5    7947   VIA6    794F   VIA7    7956
 VIA8    795F   VIA9    7968

FINE ASSEMBLER
```

SYS (4)   e quindi

M 7800 78A0

che fara' apparire sullo schermo una prima parte di memoria da modificare. Il metodo di modifica di memoria e' quello di risalire con il cursore e modificare i vecchi contenuti con quelli riportati nella lista. Ad ogni fine riga modificata convalidare con RETURN. Proseguire infine con M 78A8 7nnn, fino ad arrivare al termine in 7AC1.

Dopo aver eseguito queste operazioni, prima di eseguire la SYS (30720) e' bene salvare, sempre in modo TIM, il programma in LM con il sistema:

S "EXTRA BASIC",01,7800,7AC1

se il supporto di salvataggio e' la TAPE CASSETTA, mentre:

S "EXTRA BASIC",08,7800,7AC1

se siamo possessori di unita' disco.

Per uscire dal TIM battere la lettera X e quindi RETURN.

N.B.: Nella lista in LM tutti i dati sottolineati sono quelli che devono essere cambiati con i rispettivi valori (BASIC 3.0) posti fra parentesi nel listato in Assembler.

```
 10 I=30720: K=I
100 PRINT "[CLR][2 DOWN] ATTENDERE PREGO"
110 READ A$: IF A$ <> "-1" THEN 140
120 PRINT "[2 DOWN] ESEGUI PER ATTIVARE : SYS("K")"
130 END
140 A1$=LEFT$(A$,1)
150 A2$=RIGHT$(A$,1)
160 IF A1$ é "A" THEN 180
170 A1$=STR$(ASC(A1$)-55)
180 IF A2$ é "A" THEN 200
190 A2$=STR$(ASC(A2$)-55)
200 A=VAL(A1$)*16+VAL(A2$)
210 IF A<0 OR A>255 THEN 240
220 POKE I,A: I=I+1
230 GOTO 110
240 PRINT "BYTE "I" = "A" ???"
250 STOP
260 DATA A9,4C,85,70,A9,2D,85,71
270 DATA A9,78,85,72,A9,77,85,35
280 DATA ..,..,..,..,..,..,..,..
290 DATA ..,..,             ,..,..
300 DATA ..,.., ecc. ecc. ,..,..
310 DATA ..,..,             ,..,..
320 DATA ..,..,..,..,..,..,..,..
330 DATA ..,..,..,..,..,..,..,..
nnn DATA 76,00,-1
```

```
7800 A9 4C 85 70 A9 2D 85 71
7808 A9 78 85 72 A9 77 85 35
7810 85 31 A9 FF 85 30 85 34
7818 4C FF B3 A9 E6 85 70 A9
7820 77 85 71 A9 D0 85 72 20
7828 E3 78 20 FF B3 8E 90 03
7830 BA BD 01 01 C9 23 D0 10
7838 BD 02 01 C5 C6 F0 09 E6
7840 77 D0 02 E6 78 4C 54 78
7848 AE 90 03 E6 77 D0 02 E6
7850 78 4C 76 00 A0 00 B1 77
7858 C9 40 F0 03 4C 76 00 E6
7860 77 D0 02 E6 78 B1 77 E6
7868 77 D0 02 E6 78 C9 49 D0
7870 17 A2 80 A0 00 84 01 86
7878 02 B1 01 49 80 91 01 C8
7880 D0 F7 E8 E0 84 D0 F0 60
7888 C9 23 D0 13 20 84 BD 20
7890 2D C9 20 22 D7 98 20 22
7898 D7 D0 01 60 4C 76 00 C9
78A0 26 D0 3C 78 A9 B2 85 90
78A8 A9 78 85 91 A9 01 85 FF
78B0 58 60 A5 97 C5 FD F0 09
78B8 85 FD A9 10 85 FE 4C 55
78C0 E4 C9 FF F0 F9 A5 FE F0
78C8 04 C6 FE D0 F1 C6 FF D0
78D0 ED A9 02 85 FF A9 00 85
78D8 97 A9 02 85 A8 D0 DF C9
78E0 4B D0 1A A9 55 85 90 A9
78E8 E4 85 91 A9 10 85 FD A9
78F0 00 8D 4B E8 8D 4A E8 8D
78F8 48 E8 85 01 60 C9 53 D0
7900 18 A9 10 8D 4B E8 20 D1
7908 C8 E4 01 F0 05 86 01 8E
7910 4A E8 20 27 C9 8E 48 E8
7918 60 C9 32 D0 2A A0 28 84
7920 22 A0 00 84 20 A0 BF A2
7928 83 86 21 86 23 B1 20 91
7930 22 88 C0 FF D0 F7 CA E0
7938 7F D0 EE A0 27 A9 20 91
7940 20 88 10 FB 4C 76 00 C9
7948 38 D0 04 20 69 E3 60 C9
7950 51 D0 03 4C 1B 78 C9 41
7958 D0 05 A9 0C 8D 4C E8 C9
7960 42 D0 05 A9 0E 8D 4C E8
```

```
7968 C9 4D D0 20 18 A2 80 A0
7970 00 84 21 86 22 A5 22 E9
7978 03 85 01 A9 00 85 00 B1
7980 21 91 00 C8 D0 F9 E8 E0
7988 84 D0 E8 60 C9 4C D0 20
7990 18 A2 80 A0 00 84 21 86
7998 22 A5 22 E9 03 85 01 A9
79A0 00 85 00 B1 00 91 21 C8
79A8 D0 F9 E8 E0 84 D0 E8 60
79B0 C9 58 D0 28 18 A2 80 A0
79B8 00 84 21 86 22 A5 22 E9
79C0 03 85 01 A9 00 85 00 B1
79C8 21 85 02 B1 00 91 21 A5
79D0 02 91 00 C8 D0 F1 E8 E0
79D8 84 D0 E0 60 C9 50 F0 03
79E0 4C 79 7A 20 D1 C8 E0 03
79E8 30 03 4C 76 00 8E 7A 02
79F0 A9 04 85 D2 A9 FF 85 D3
79F8 A9 04 85 D4 A2 04 20 63
7A00 F5 A2 04 20 C9 FF 20 0D
7A08 7A 20 E7 FF 60 A9 80 A0
7A10 00 84 21 85 22 A2 00 B1
7A18 21 20 72 7A C9 40 30 24
7A20 C9 7E 30 25 C9 80 30 1C
7A28 C9 BF 30 22 C9 FF F0 23
7A30 9D 7B 02 E8 E0 28 F0 20
7A38 C8 D0 DC E6 22 A5 22 C9
7A40 84 D0 D4 60 69 40 4C 30
7A48 7A 69 80 4C 30 7A E9 40
7A50 4C 30 7A A9 BF 4C 30 7A
7A58 A9 0D 9D 7B 02 E8 A9 00
7A60 9D 7B 02 84 00 A9 7A A0
7A68 02 20 1D BB A4 00 C8 4C
7A70 15 7A C9 80 30 02 E9 80
7A78 60 C9 43 F0 03 4C 76 00
7A80 20 D1 C8 86 21 A9 00 85
7A88 00 A9 80 85 01 A2 17 A0
7A90 00 A5 21 91 00 C8 C0 28
7A98 D0 F9 E0 17 D0 21 A0 00
7AA0 18 A5 00 69 28 85 00 90
7AA8 02 E6 01 A5 21 91 00 A0
7AB0 27 91 00 CA D0 E8 A5 00
7AB8 69 28 85 00 4C 8F 7A 4C
7AC0 76 00
```

```
1 REM ***** SPOSTA SU TV (4-6-8-2 1-3-9-7) *****
10 PRINT"⊐"
20 A=500
30 POKE32767+A,81
40 IFPEEK(151)=42THENPOKE32767+A,32:A=A-1
41 IFPEEK(151)=26THENPOKE32767+A,32:A=A+39
42 IFPEEK(151)=25THENPOKE32767+A,32:A=A+41
43 IFPEEK(151)=58THENPOKE32767+A,32:A=A-41
44 IFPEEK(151)=57THENPOKE32767+A,32:A=A-39
45 IFPEEK(151)=50THENPOKE32767+A,32:A=A-40
50 IFPEEK(151)=41THENPOKE32767+A,32:A=A+1
55 IFPEEK(151)=18THENPOKE32767+A,32:A=A+40
60 IFA>1000THENA=A-1000
65 IFA<0THENA=A+1000
70 GOTO30
```

BASIC 4.0 MEMORY MAP

Compiled by Jim Butterfield

There are some differences between usage between the 40- and 80-column machines.

Hex	Decimal	Description
0000-0002	0-2	USR jump
0003	3	Search character
0004	4	Scan-between-quotes flag
0005	5	Input buffer pointer; # of subscripts
0006	6	Default DIM flag
0007	7	Type: FF=string, 00=numeric
0008	8	Type: 80=integer, 00=floating point
0009	9	Flag: DATA scan; LIST quote; memory
000A	10	Subscript flag; FNX flag
000B	11	0=INPUT; $40=GET; $98=READ
000C	12	ATN sign/Comparison Evaluation flag
000D-000F	13-15	Disk status DS$ descriptor
0010	16	Current I/O device for prompt-suppress
0011-0012	17-18	Integer value (for SYS, GOTO etc)
0013-0015	19-21	Pointers for descriptor stack
0016-001E	22-30	Descriptor stack(temp strings)
001F-0022	31-34	Utility pointer area
0023-0027	35-39	Product area for multiplication
0028-0029	40-41	Pointer: Start-of-Basic
002A-002B	42-43	Pointer: Start-of-Variables
002C-002D	44-45	Pointer: Start-of-Arrays
002E-002F	46-47	Pointer: End-of-Arrays
0030-0031	48-49	Pointer: String-storage(moving down)
0032-0033	50-51	Utility string pointer
0034-0035	52-53	Pointer: Limit-of-memory
0036-0037	54-55	Current Basic line number
0038-0039	56-57	Previous Basic line number
003A-003B	58-59	Pointer: Basic statement for CONT
003C-003D	60-61	Current DATA line number
003E-003F	62-63	Current DATA address
0040-0041	64-65	Input vector
0042-0043	66-67	Current variable name
0044-0045	68-69	Current variable address
0046-0047	70-71	Variable pointer for FOR/NEXT
0048-0049	72-73	Y-save; op-save; Basic pointer save
004A	74	Comparison symbol accumulator
004B-0050	75-80	Misc work area, pointers, etc
0051-0053	81-83	Jump vector for functions
0054-005D	84-93	Misc numeric work area
005E	94	Accum#1: Exponent
005F-0062	95-98	Accum#1: Mantissa
0063	99	Accum#1: Sign
0064	100	Series evaluation constant pointer
0065	101	Accum#1 hi-order (overflow)
0066-006B	102-107	Accum#2: Exponent, etc.
006C	108	Sign comparison, Acc#1 vs #2
006D	106	Accum#1 lo-order (rounding)
006E-006F	110-111	Cassette buff len/Series pointer
0070-0087	112-135	CHRGET subroutine; get Basic char
0077-0078	119-120	Basic pointer (within subrtn)
0088-008C	136-140	Random number seed.
008D-008F	141-143	Jiffy clock for TI and TI$
0090-0091	144-145	Hardware interrupt vector
0092-0093	146-147	BRK interrupt vector
0094-0095	148-149	NMI interrupt vector
0096	150	Status word ST
0097	151	Which key down; 255=no key
0098	152	Shift key: 1 if depressed
0099-009A	153-154	Correction clock
009B	155	Keyswitch PIA: STOP and RVS flags
009C	156	Timing constant for tape
009D	157	Load=0, Verify=1
009E	158	Number of characters in keybd buffer
009F	159	Screen reverse flag
00A0	160	IEEE output; 255=character pending
00A1	161	End-of-line-for-input pointer
00A3-00A4	163-164	Cursor log (row, column)
00A5	165	IEEE output buffer
00A6	166	Key image

00A7	167	0=flash cursor
00A8	168	Cursor timing countdown
00A9	169	Character under cursor
00AA	170	Cursor in blink phase
00AB	171	EOT received from tape
00AC	172	Input from screen/from keyboard
00AD	173	X save
00AE	174	How many open files
00AF	175	Input device, normally 0
00B0	176	Output CMD device, normally 3
00B1	177	Tape character parity
00B2	178	Byte received flag
00B3	179	Logical Address temporary save
00B4	180	Tape buffer character; MLM command
00B5	181	File name pointer; MLM flag, counter
00B7	183	Serial bit count
00B9	185	Cycle counter
00BA	186	Tape writer countdown
00BB-00BC	187-188	Tape buffer pointers, #1 and #2
00BD	189	Write leader count; read pass1/2
00BE	190	Write new byte; read error flag
00BF	191	Write start bit; read bit seq error
00C0-00C1	192-193	Error log pointers, pass1/2
00C2	194	0=Scan/1-15=Count/$40=Load/$80=End
00C3	195	Write leader length; read checksum
00C4-00C5	196-197	Pointer to screen line
00C6	198	Position of cursor on above line
00C7-00C8	199-200	Utility pointer: tape, scroll
00C9-00CA	201-202	Tape end addrs/End of current program
00CB-00CC	203-204	Tape timing constants
00CD	205	0=direct cursor, else programmed
00CE	206	Tape read timer 1 enabled
00CF	207	EOT received from tape
00D0	208	Read character error
00D1	209	# characters in file name
00D2	210	Current file logical address
00D3	211	Current file secondary addrs
00D4	212	Current file device number
00D5	213	Right-hand window or line margin
00D6-00D7	214-215	Pointer: Start of tape buffer
00D8	216	Line where cursor lives
00D9	217	Last key/checksum/misc.
00DA-00DB	218-219	File name pointer
00DC	220	Number of INSERTs outstanding
00DD	221	Write shift word/read character in
00DE	222	Tape blocks remaining to write/read
00DF	223	Serial word buffer
00E0-00F8	224-248	(40-column) Screen line wrap table
00E0-00E1	224-225	(80-column) Top, bottom of window
00E2	226	(80-column) Left window margin
00E3	227	(80-column) Limit of keybd buffer
00E4	228	(80-column) Key repeat flag
00E5	229	(80-column) Repeat countdown
00E6	230	(80-column) New key marker
00E7	231	(80-column) Chime time
00E8	232	(80-column) HOME count
00E9-00EA	233-234	(80-column) Input vector
00EB-00EC	235-236	(80-column) Output vector
00F9-00FA	249-250	Cassette status, #1 and #2
00FB-00FC	251-252	MLM pointer/Tape start address
00FD-00FE	253-254	MLM, DOS pointer, misc.
0100-010A	256-266	STR$ work area, MLM work
0100-013E	256-318	Tape read error log
0100-01FF	256-511	Processor stack
0200-0250	512-592	MLM work area; Input buffer
0251-025A	593-602	File logical address table
025B-0264	603-612	File device number table
0265-026E	613-622	File secondary adds table
026F-0278	623-632	Keyboard input buffer
027A-0339	634-825	Tape#1 input buffer
033A-03F9	826-1017	Tape#2 input buffer
033A	826	DOS character pointer
033B	827	DOS drive 1 flag
033C	828	DOS drive 2 flag
033D	829	DOS length/write flag
033E	830	DOS syntax flags

```
033F-0340    831-832    DOS disk ID
0341         833        DOS command string count
0342-0352    834-850    DOS file name buffer
0353-0380    851-896    DOS command string buffer
03EE-03F7   1006-1015   (80-column) Tab stop table
03FA-03FB   1018-1019   Monitor extension vector
03FC        1020        IEEE timeout defeat
0400-7FFF   1024-32767  Available RAM including expansion
8000-83FF  32768-33791  (40-column) Video RAM
8000-87FF  32768-34815  (80-column) Video RAM
9000-AFFF  36864-45055  Available ROM expansion area
B000-DFFF  45056-57343  Basic, DOS, Machine Lang Monitor
E000-E7FF  57344-59391  Screen, Keyboard, Interrupt programs
E810-E813  59408-59411  PIA 1 - Keyboard I/O
E820-E823  59424-59427  PIA 2 - IEEE-488 I/0
E840-E84F  59456-59471  VIA - I/O and timers
E880-E881  59520-59521  (80-column) CRT Controller
F000-FFFF  61440-65535  Reset, I/O handlers, Tape routines
```

PET 4.0 ROM ROUTINES

Jim Butterfield

Toronto

The 40-character and 80-character machines are the same except for addresses $E000-$E7FF.

This map shows where various routines lie. The first address is not necessarily the proper entry point for the routine. Similarly, many routines require register setup or data preparation before calling.

```
                    Description
B000-B065 Action addresses for primary keywords
B066-B093 Action addresses for functions
B094-B0B1 Hierarchy and action addresses for operators
B0B2-B20C Table of Basic keywords
B20D-B321 Basic messages, mostly error messages
B322-B34F Search the stack for FOR or GOSUB activity
B350-B392 Open up space in memory
B393-B39F Test: stack too deep?
B3A0-B3CC Check available memory
B3CD      Send canned error message, then:
B3FF-B41E Warm start; wait for Basic command
B41F-B4B5 Handle new Basic line input
B4B6-B4E1 Rebuild chaining of Basic lines
B4E2-B4FA Receive line from keyboard
B4FB-B5A2 Crunch keywords into Basic tokens
B5A3-B5D1 Search Basic for given line number
B5D2      Perform NEW, and;
B5EC-B621 Perform CLR
B622-B62F Reset Basic execution to start
B630-B6DD Perform LIST
B6DE-B784 Perform FOR
B785-B7B6 Execute Basic statement
B7B7-B7C5 Perform RESTORE
B7C6-B7ED Perform STOP or END
B7EE-B807 Perform CONT
B808-B812 Perform RUN
B813-B82F Perform GOSUB
B830-B85C Perform GOTO
B85D      Perform RETURN, then:
B883-B890 Perform DATA: skip statement
B891      Scan for next Basic statement
B894-B8B2 Scan for next Basic line
B8B3      Perform IF, and perhaps:
B8C6-B8D5 Perform REM: skip line
B8D6-B8F5 Perform ON
B8F6-B92F Accept fixed-point number
B930-BA87 Perform LET
BA88-BA8D Perform PRINT#
BA8E-BAA1 Perform CMD
BAA2-BB1C Perform PRINT
BB1D-BB39 Print string from memory
```

```
BB3A-BB4B Print single format character
BB4C-BB79 Handle bad input data
BB7A-BBA3 Perform GET
BBA4-BBBD Perform INPUT#
BBBE-BBF4 Perform INPUT
BBF5-BC01 Prompt and receive input
BC02-BCF6 Perform READ
BCF7-BD18 Canned Input error messages
BD19-BD71 Perform NEXT
BD72-BD97 Check type mismatch
BD98      Evaluate expression
BEE9      Evaluate expression within parentheses
BEEF      Check parenthesis, comma
BF00-BF0B Syntax error exit
BF8C-C046 Variable name setup
C047-C085 Set up function references
C086-C0B5 Perform OR, AND
C0B6-C11D Perform comparisons
C11E-C12A Perform DIM
C12B-C1BF Search for variable
C1C0-C2C7 Create new variable
C2C8-C2D8 Setup array pointer
C2D9-C2DC 32768 in floating binary
C2DD-C2FB Evaluate integer expression
C2FC-C4A7 Find or make array
C4A8      Perform FRE, and:
C4BC-C4C8 Convert fixed-to-floating
C4C9-C4CE Perform POS
C4CF-C4DB Check not Direct
C4DC-C509 Perform DEF
C50A-C51C Check FNx syntax
C51D-C58D Evaluate FNx
C58E-C59D Perform STR$
C59E-C5AF Do string vector
C5B0-C61C Scan, set up string
C61D-C669 Allocate space for string
C66A-C74E Garbage collection
C74F-C78B Concatenate
C78C-C7B4 Store string
C7B5-C810 Discard unwanted string
C811-C821 Clean descriptor stack
C822-C835 Perform CHR$
C836-C861 Perform LEFT$
C862-C86C Perform RIGHT$
C86D-C896 Perform MID$
C897-C8B1 Pull string data
C8B2-C8B7 Perform LEN
C8B8-C8C0 Switch string to numeric
C8C1-C8D0 Perform ASC
C8D1-C8E2 Get byte parameter
C8E3-C920 Perform VAL
C921-C92C Get two parameters for POKE or WAIT
C92D-C942 Convert floating-to-fixed
C943-C959 Perform PEEK
C95A-C962 Perform POKE
C963-C97E Perform WAIT
C97F-C985 Add 0.5
C986      Perform subtraction
C998-CA7C Perform addition
CA7D-CAB3 Complement accum#1
CAB4-CAB8 Overflow exit
CAB9-CAF1 Multiply-a-byte
CAF2-CB1F Constants
CB20      Perform LOG
CB5E-CBC1 Perform multiplication
CBC2-CBEC Unpack memory into accum#2
CBED-CC09 Test & adjust accumulators
CC0A-CC17 Handle overflow and underflow
CC18-CC2E Multiply by 10
CC2F-CC33 10 in floating binary
CC34      Divide by 10
CC3D      Perform divide-by
CC45-CCD7 Perform divide-into
CCD8-CCFC Unpack memory into accum#1
CCFD-CD31 Pack accum#1 into memory
CD32-CD41 Move accum#2 to #1
```

```
CD42-CD50 Move accum#1 to #2
CD51-CD60 Round accum#1
CD61-CD6E Get accum#1 sign
CD6F-CD8D Perform SGN
CD8E-CD90 Perform ABS
CD91-CDD0 Compare accum#1 to memory
CDD1-CE01 Floating-to-fixed
CE02-CE28 Perform INT
CE29-CEB3 Convert string to floating-point
CEB4-CEE8 Get new ASCII digit
CEE9-CEF8 Constants
CF78      Print IN, then:
CF7F-CF92 Print Basic line #
CF93-D0C6 Convert floating-point to ASCII
D0C7-D107 Constants
D108      Perform SQR
D112      Perform power function
D14B-D155 Perform negation
D156-D183 Constants
D184-D1D6 Perform EXP
D1D7-D220 Series evaluation
D221-D228 RND constants
D229-D281 Perform RND
D282      Perform COS
D289-D2D1 Perform SIN
D2D2-D2FD Perform TAN
D2FE-D32B Constants
D32C-D35B Perform ATN
D35C-D398 Constants
D399-D3B5 CHRGET sub for zero page
D3B6-D471 Basic cold start
D472-D716 Machine Language Monitor
D717-D7AB MLM subroutines
D7AC-D802 Perform RECORD
D803-D837 Disk parameter checks
D838-D872 Dummy disk control messages
D873-D919 Perform CATALOG or DIRECTORY
D91A-D92E Output
D92F-D941 Find spare secondary address
D942-D976 Perform DOPEN
D977-D990 Perform APPEND
D991-D9D1 Get disk status
D9D2-DA06 Perform HEADER
DA07-DA30 Perform DCLOSE
DA31-DA64 Set up disk record
DA65-DA7D Perform COLLECT
DA7E-DAA6 Perform BACKUP
DAA7-DAC6 Perform COPY
DAC7-DAD3 Perform CONCAT
DAD4-DB0C Insert command string values
DB0D-DB39 Perform DSAVE
DB3A-DB65 Perform DLOAD
DB66-DB98 Perform SCRATCH
DB99-DB9D Check Direct command
DB9E-DBD6 Query ARE YOU SURE?
DBD7-DBE0 Print BAD DISK
DBE1-DBF9 Clear DS$ and ST
DBFA-DC67 Assemble disk command string
DC68-DE29 Parse Basic DOS command
DE2C-DE48 Get Device number
DE49-DE86 Get file name
DE87-DE9C Get small variable parameter
** Entry points only for E000-E7FF **
E000      Register/screen initialization
E0A7      Input from keyboard
E116      Input from screen
E202      Output character
E442      Main Interrupt entry
E455      Interrupt: clock, cursor, keyboard
E600      Exit from Interrupt
**                                   **
F000-F0D1 File messages
F0D2      Send 'Talk'
F0D5      Send 'Listen'
F0D7      Send IEEE command character
F109-F142 Send byte to IEEE
```

```
F143-F150 Send byte and clear ATN
F151-F16B Option: timeout or wait
F16C-F16F DEVICE NOT PRESENT
F170-F184 Timeout on read, clear control lines
F185-F192 Send canned file message
F193-F19D Send byte, clear control lines
F19E-F1AD Send normal (deferred) IEEE char
F1AE-F1BF Drop IEEE device
F1C0-F204 Input byte from IEEE
F205-F214 GET a byte
F215-F265 INPUT a byte
F266-F2A1 Output a byte
F2A2      Abort files
F2A6-F2C0 Restore default I/O devices
F2C1-F2DC Find/setup file data
F2DD-F334 Perform CLOSE
F335-F342 Test STOP key
F343-F348 Action STOP key
F349-F350 Send message if Direct mode
F351-F355 Test if Direct mode
F356-F400 Program load subroutine
F401-F448 Perform LOAD
F449-F46C Print SEARCHING
F46D-F47C Print LOADING or VERIFYING
F47D-F4A4 Get Load/Save parameters
F4A5-F4D2 Send name to IEEE
F4D3-F4F5 Find specific tape header
F4F6-F50C Perform VERIFY
F50D-F55F Get Open/Close parameters
F560-F5E4 Perform OPEN
F5E5-F618 Find any tape header
F619-F67A Write tape header
F67B-F694 Get start/end addrs from header
F695-F6AA Set buffer address
F6AB-F6C2 Set buffer start & end addrs
F6C3-F6CB Perform SYS
F6CC-F6DC Set tape write start & end
F6DD-F767 Perform SAVE
F768-F7AE Update clock
F7AF-F7FD Connect input device
F7FE-F84A Connect output device
F84B-F856 Bump tape buffer pointer
F857-F879 Wait for PLAY
F87A-F88B Test cassette switch
F88C-F899 Wait for RECORD
F89A      Initiate tape read
F8CB      Initiate tape write
F8E0-F92A Common tape I/O
F92B-F934 Test I/0 complete
F935-F944 Test STOP key
F945-F975 Tape bit timing adjust
F976-FA9B Read tape bits
FA9C-FBBA Read tape characters
FBBB-FBC3 Reset tape read address
FBC4-FBC8 Flag error into ST
FBC9-FBD7 Reset counters for new byte
FBD8-FBF3 Write a bit to tape
FBF4-FC85 Tape write
FC86-FCBF Write tape leader
FCC0-FCDA Terminate tape; restore interrupt
FCDB-FCEA Set interrupt vector
FCEB-FCF8 Turn off tape motor
FCF9-FD0A Checksum calculation
FD0B-FD15 Advance load/save pointer
FD16-FD4B Power-on Reset
FD4C-FD5C Table of interrupt vectors
** Jump table:                    **
FF93-FF9E CONCAT,DOPEN,DCLOSE,RECORD
FF9F-FFAA HEADER,COLLECT,BACKUP,COPY
FFAB-FFB6 APPEND,DSAVE,DLOAD,CATALOG
FFB7-FFBC RENAME,SCRATCH
FFBD      Get disk status
FFC0      OPEN
FFC3      CLOSE
FFC6      Set input device
FFC9      Set output device
```

```
FFCC          Restore default I/O devices
FFCF          INPUT a byte
FFD2          Output a byte
FFD5          LOAD
FFD8          SAVE
FFDB          VERIFY
FFDE          SYS
FFE1          Test stop key
FFE4          GET byte
FFE7          Abort all files
FFEA          Update clock
FFFA-FFFF     Hard vectors:  NMI, Reset, INT
```

Pensiamo di aver fatto cosa utile riportando per intero la mappa di memoria del BASIC versione 4.0 valida per i PET-CBM modelli della serie 4000 e 8000, nonche' per la serie 3000 a cui sono state cambiate le ROM.
Abbiamo pensato di riportare in originale tale Mappa in quanto le descrizioni riportate in lingua inglese sono estremamente chiare sia per chi si accinge da poco al linguaggio macchina, e tanto meno per chi e' gia un po' smaliziato.
Un sentito ringraziamento a Jim Butterfield che ha redatto con somma maestria e inimitabile esperienza questa tabella di estrema utilita'.

A PROPOSITO DI "RAGNO NERO"....

Ringraziamo quei lettori che ci hanno fatto notare che il programma, sul loro PET, non girava a dovere.
RAGNO NERO, infatti, e' stato creato per il BASIC 3.0 nuove ROM, e necessita di alcune correzioni per poter funzionare sul BASIC 4.0 o con le vecchie ROM del BASIC versione 2.0.
Ecco le modifiche da apportare al programma per le varie versioni di BASIC.

Vecchie ROM:

Oltre alle correzioni specificate nell'articolo, sostituire come segue:

```
POKE 216  --è POKE 245
POKE 198  --è POKE 226
SYS 57949 --è SYS 58843
```

BASIC 4.0:

Sostituire SYS 57949 con SYS 57471

A questo punto non ci sono piu' problemi. Consiglio di copiare fedelmente, almeno la prima volta, le righe di intestazione 50-80, poiche' inizialmente il programma si basa proprio su queste righe per il movimento del ragno e per il suo deprecabile comportamento.

uguale o simile

di Gloriano Rossi

In alcuni casi, durante le elaborazioni di dati, e' necessario poter paragonare una determinata stringa di caratteri con altre gia' residenti in memoria.

Questa 'faccenda' potrebbe sembrare, in una prima analisi, molto semplice.

Prendo la variabile X$ e la confronto con n variabili fino a che non trovo quella uguale.
Tutto cio' e' in parte vero. Infatti se la variabile X$ risulta essere piu' corta rispetto a quella che eventualmente potrebbe essere uguali, il BASIC, non ne riconosce l'ugualglianza.

```
100 REM**********************************
110 REM* ROUTINE UGUALE E SIMILE        *
120 REM**********************************

130 REM**********************************
140 REM* DI GLORIANO ROSSI      I2KH   *
150 REM*        POCKET GROUP            *
160 REM**********************************
170 N=9:DIMA$(N),A(N)
180 A$(1)="GLORIANO  ROSSI"
190 A$(2)="ROBERTO SOZZANI"
200 A$(3)="MASSIMO ROSSI"
210 A$(4)="PIPPO POPPOLINI"
220 A$(5)="PIPPO FILIPPINI"
230 A$(6)="BRUNO BRAZZODURO"
240 A$(7)="FIORENZO"
250 A$(8)="GIOVANNI"
260 A$(9)="GIORGIO"
265 REM"

270 INPUT"QUALE NOME  ";X$
```

La prima parte della routine proposta ("Riconosce l'uguaglianza") serve proprio per riconoscere l'uguaglianza fra' la stringa X$ (riga 270) e la tabella A$(n).

```
280 REM**********************************
290 REM* RICONOSCE L'EGUAGLIANZA        *
300 REM**********************************
310 FORI=1TON
320 IFLEFT$(A$(I),LEN(X$))=X$THENPRINTA$(I):I=N+1:P=1
330 NEXT
340 IFP=1THEN450
```

In riga 320 viene esaminata la parte dell'elemento della tabella A$(n) per una dimensione pari alla lunghezza uguale a quella di X$.

Quando si e' trovato che esiste la condizione di ugualglianza si interviene eseguendo la visualizzazione e quindi si esegue una forzata uscita dal loop (I=I+1).
Il programmino riportato e' stato congegnato in maniera tale che se non si verifica l'uguaglianza ricercata va ad eseguire la seconda parte della routine, per ovviare cio', in caso di avvenuto accoppiamento, si forza il numero 1 nella variabile P. Cio' fatto, in riga 340, se P sara' uguale ad 1 si uscira' dalla routine.

Nel caso contrario, cioe' nessun accoppiamnto e' stato possibile, si prosegue con la seconda parte del programma.

Ecco quindi che si puo' presentare l'oppurtunita' di confrontare il contenuto della variabile X$ sempre con gli elementi della tabella A$(n) con lo scopo di individuare quale elemento e' piu' simile a quello noto.

Per risolvere questo problemino viene presentata la seconda parte della routine in oggetto: "Riconosce il piu' simile".

```
350 REM**********************************
360 REM* RICONOSCE IL PIU' SIMILE       *
370 REM**********************************
380 FORR=1TON
390 FORI=1TOLEN(X$)
400 IFMID$(A$(R),I,1)=MID$(X$,I,1)THENA(R)=A(R)+1
410 NEXT:NEXT
420 FORI=1TON
430 IFA(I)>A(X)THENX=I
440 NEXT:PRINTA$(X)
450 END
```

L'esame viene eseguito chiaramente su tutti gli elementi della tabella A$(n).

Carattere per carattere viene confrontato ogni elemento della tabella A$(n) e se si incontra una uguaglianza di carattere si incrementa l'elemento corrispondente della tabella numerica A(n).

Questo procedimento viene eseguito per ogni carattere di ogni elemento della tabella, ogni volta per una lunghezza pari alla lunghezza della variabile X$.

Terminato questo tipo di esame, si controlla quale elemento ha meritato piu' "punti", e proprio questo elemento sara' oggetto della evidenziazione in edit di schermo.

L'attualizzazione di questa routine potrebbe trovare numerose applicazioni, una delle quali potrebbe essere quella di sviluppare un gioco tipo "E' arrivato un bastimento carico, carico di....".
Si dovra' incrementare ad ogni risposta la tabella A$(n) con il nuovo elemento riconosciuto disuguale. E' chiaro che sia la tabella A$(n) e A(n) dovranno essere opportunamente definite con una DIM in quanto, si sa, quando gli elementi superano il numero di 10 occorre ricorrere a questa facilita' BASIC.

Provate, divertitevi e... mandate tranquillamente i vostri elaborati. Il migliore verra' certamente pubblicato su POCKET PET.

Cosa c'é dietro il BASIC

di Massimo Rossi

Vi siete mai chiesti, che cosa succede, quando premete il tasto di RETURN, dopo aver scritto un comando, o una riga di programma?. Forse non vi sembrera' possibile, ma si potrebbe parlare per ore di questo argomento, e pensiamo che anche un solo accenno superficiale ad esso, possa essere interessante per tutti gli utilizzatori del PET.

Vediamo insieme, percio', che cosa avviene, quando premiamo il fatidico tasto di RETURN.

Vi sono due possibilita' differenti: se all'inizio della riga abbiamo scritto un numero, questo verra' considerato come inizio della riga del BASIC, e la riga verra' immagazzinata come facente parte di un programma. Per inciso, questo numero, deve essere minore di 65536, altrimenti avremo un messaggio di SYNTAX ERROR; la ragione di cio', e' che il numero di riga, e' immagazzinato in due locazioni successive di un Byte ciascuna, e percio' potremo scrivere numeri compresi tra 0 e il massimo numero di combinazioni possibili tra i due Bytes di che possono contenere un massimo di 255 numeri ciascuno. Percio': 255 * 255 = 65535 che e' il piu' alto numero di riga utilizzabile.

Se, invece, non vi e' alcun numero all'inizio di una riga, avremo un comando diretto, che verra' eseguito, non appena verra' premuto il RETURN. Ovviamente, quanto scritto, non verra' immagazinato in memoria, ma verra' perso, se cancelleremo il video, senza aver prima premuto il RETURN, sulla riga che ci interessava, dopo aver aggiunto il numero al suo inizio.

In tutti e due i casi precedenti, il PET, una volta lette le parole in BASIC, sullo schermo, o nella memoria, le deve tradurre in un linguaggio comprensibile al microprocessore, cioe' in linguaggio mcchina.

Questo e' un linguaggio scritto in codice binario, dove esistono solo numeri, che pero' possono significare anche lettere o istruzioni, secondo un preciso codice che e' la lingua del microprocesore 6502.

Questa lingua, comunica al microprocessore di eseguire tante piccole operazioni elementari, che messe tutte insieme, possono eseguire le stesse istruzioni, che noi diamo in BASIC. Cio'. significa che quando noi scriviamo, per esempio, PRINT, il PET lo riconosce, e va a cercare nella sua memoria non cancellabile (ROM), le routines, precostituite, che interpretano il comando BASIC, traducendolo in tante piccole operazioni in linguaggio macchina. Questo procedimento, viene ripetuto per ogni parola BASIC, che viene incontrata, fino ad aver eseguito tutto il programma o l'istruzione.

Ovviamente queste istruzioni in linguaggio macchina, essendo scritte su ROM, hanno bisogno di un'area dove possono immagazzinare le variabili e i dati che il sistema operativo usa per eseguire tutte le sue funzioni: questo e' compito delle prime 1023 celle di memoria, che, ovviamente, non vengono mai coperte da programmi BASIC.

Possiamo immaginare, quindi, dove il computer possa mettere la prima istruzione di BASIC: nella prima locazione di memoria libera, cioe' la 1024.

In realta', nella 1024, c'e sempre uno 0, poiche' questo significa, per il computer, che li' inizia il BASIC. Come abbiamo detto prima, ogni cella di memoria, puo' contenere solo dei numeri, e , poiche' le istruzioni del BASIC, sono un numero finito di parole chiave, i tecnici progettisti, hanno pensato che si sarebbe potuto associare un numero ad ogni parola BASIC, in modo da costruire una tabella interna di codifica.

Questo permette di usare un solo numero, per indicare una parola BASIC, che altrimenti dovrebbe occupare tante celle di memoria, quante sono le lettere che la compongono.

Facciamo un esempio: se una istruzione contiene la parola PRINT, noi dovremmo memorizzarla come P,R,I,N,T, cioe' ogni lettera occuperebbe un Byte, o cella di memoria, cioe' un totale di 5 Bytes. Se invece ricorressimo ad una tabella di abbreviazioni, scriveremmo il numero corrispondente a PRINT, che e' 99, occupando una sola cella di memoria.

Si comprende da questo esempio, come si possano risparmiare enormi quantita' di memoria, usando dei numeri in codice per immagazzinare i programmi in BASIC.

Ora proviamo ad immaginare come si potrebbe fare a scrivere delle intere istruzioni, nella memoria.

Prima di tutto, dobbiamo segnalare che ci troviamo all'inizio di una istruzione. Questo lo facciamo, con un numero che certamente, non fa parte dellla tabella dei comandi BASIC: lo Zero. Questo viene posto come segnale di fine istruzione (inteso come termine di riga BASIC, in quanto si possono porre piu' istruzioni, sulla stessa riga, intervallate da dei ":"), o come segnale di inizio istruzione, che e' poi la stessa cosa.

Riassumendo: ogni volta che incontriamo uno 0, nella memoria dedicata al BASIC, sappiamo che essa divide due istruzioni.

A questo punto, dobbiamo dire al computer, la lunghezza dell'istruzione, questo per facilitare il meccanismo di LIST. Questo viene fatto, indicando non tanto la lunghezza della istruzione, ma per maggior comodita', l'indirizzo (cioe' la posizione come numero di cella di memoria) in cui si trova la successiva istruzione.

Questa informazione, deve contenere un numero che puo' anche essere di cinque cifre, e percio' essere piu' grande di 255 (il numero massimo contenibile in un Byte): percio' dovremo usare due Bytes per questo scopo.

Ora, possiamo mettere il numero di linea, che, ovviamente, sara' in esadecimale, e occupera' anch'esso due Bytes, come abbiamo detto precedentemente.

Abbiamo percio', descritto un'istruzione tipo, avendo essa lo zero all'inizio, l'indirizzo di collegamento di due Bytes, e il numero di linea, di altrettanti Bytes. Seguira' il testo, che finira' con uno 0, dopo l'ultima istruzione, o dato della riga.

A questo punto, dobbiamo precisare alcune caratteristiche del testo BASIC: se noi scriviamo un testo tutto di seguito, come potremo aggiungere le righe intermedie? Questo e' un duro compito, assolto mirabilmente dal sistema operativo, che, ogni volta che una istruzione viene inserita nel testo, sposta tutte le altre, per creare uno spazio, e aggiorna tutti gli indirizzi di collegamento, contenuti nel testo BASIC.

Un'altra interessante caratteristica, e' il limite dell'area BASIC. Essa, infatti, si ferma ad una locazione di memoria precisa, che e' contenuta nelle locazioni di pagina zero (quella delle variabili di sistema operativo), 52 e 53 (nelle vecchie ROM 134 e 135). Quando il BASIC, o le variabili, raggiungono questa locazione, un OUT OF MEMORY ERROR, appare sullo schermo.

La presenza di queste locazioni in pagina zero, e' molto utile, se vogliamo fermare il BASIC, affinche' non copra programmi in codice macchina, che risiedono nelle locazioni piu' alte della memoria.

Bastera' fare una POKE dell'indirizzo desiderato, nelle due locazioni suddette, per limitare il BASIC, fino al punto di memoria che si desidera.

A questo punto, potra' essere interessante mettere in pratica le nozioni ora esposte, creando qualche programma di utility. Infatti, conoscendo i valori corrispondenti ai varii comandi, e il modo in cui le istruzioni vengono immagazzinate, si possono creare dei programmi in BASIC, che modificano il testo stesso.

Esistono gia' dei programmi di rinumerazione righe, scritti in BASIC, ma noi vorremmo proporvi qualcosa di nuovo, che ha un valore piu' che altro esemplificativo, ma che potrebbe tornarvi utile in certi casi, magari con qualche adattamento o miglioramento.

Si tratta di un paio di subroutines, che permettono a chi ha una stampante, di trasformare un programma che scrive su schermo, in uno per stampante. La prima routine, infatti, trasforma, una volta richiamata, il programma principale in modo che stampi su stampante, la seconda ritorna il tutto nella forma originale per lo schermo.

Tutti gli statements di PRINT, seguiti da due spazii vuoti, diventano PRINT#4, il comando per la stampa diretta. Sara' bene ricordare di scrivere i simboli di movimento cursore, in un PRINT subito precedente, non seguito da alcuno spazio vuoto, in modo che non venga tradotto su stampante come cambio carattere, o altro.

Con la seconda routine tutto ritorna come prima, e si chiude il file con la stampante.

Tutto questo e' ottenuto leggendo i valori delle locazioni di BASIC, e quando questi risultano uguali a quelli che vogliamo cambiare, vengono "pokkati" i valori relativi alle nuove istruzioni.

In riga 10000 e 11000 il valore 5000, si riferisce alla piu' alta locazione del testo BASIC in questione: potremo aumentarlo o diminuirlo a piacere, se vogliamo che la trasformazione riguardi solo una parte del programma.

Questa e' solo una idea, ma pensiamo che basti per dare un'impressione di quante siano le possibilita' di divertirsi a modificando il BASIC a piacere ...e .... buon divertimento!.

DA PRINT A PRINT#4

```
10000 FORI=1024TO5000
10010 IFPEEK(I)=153ANDPEEK(I+1)=32THEN10030
10020 NEXTI:OPEN4,4:CMD4:RETURN
10030 POKEI,152:POKEI+1,52:POKEI+2,44:NEXT
```

DA PRINT#4 A PRINT

```
11000 FORI=1024TO5000
11010 IFPEEK(I)=152ANDPEEK(I+1)=52THEN11030
11020 NEXTI:CLOSE4:RETURN
11030 POKEI,153:POKEI+1,32:POKEI+2,32:NEXT
```

di Riccardo Scotti

L'idea di scrivere questo programmino mi e' venuta leggendo il libro di Johannes Mario Simmel: "Il codice Cesare".

Questo codice era molto diffuso tra i servizi segreti per la sua relativa semplicita' e grande affidabilita'.

Infatti la codifica e la decodifica puo' essere eseguita senza l'ausilio di cifrari particolari o di tabelle tortuose. Ma l'uso di una penna ed un po' di carta puo' essere il valido mezzo per risolvere gli enigmi, cio' premesso se si conosce la frase chiave.

Il metodo e' quello della sostituzione.

Si prende una citazione come ad esempio:

COMMODOREBUSINESSMACHINE

Disponiamo sotto le lettere i numeri progressivi corrispondenti alle lettera dell'alfabeto. Sotto la A di MACHINE il numero 1, sotto la B di BUSINESS il numero 2, e cosi' di seguito fino al termine della frase. Se si incontrano delle lettere uguali si assegna, da sinistra verso destra, un numero progressivo.

Al termine di questa operazione la frase scelta si presentera' cosi':

C	O	M	M	O	D	O	R	E	B	U	S	I	N	E	S	S	M	A	C	H	I	N	E
3	17	12	13	18	5	19	20	6	2	24	21	10	15	7	22	23	14	1	4	9	11	16	8

Ora scriviamo sotto i numeri una frase da codificare:

C	O	M	M	O	D	O	R	E	B	U	S	I	N	E	S	S	M	A	C	H	I	N	E
3	17	12	13	18	5	19	20	6	2	24	21	10	15	7	22	23	14	1	4	9	11	16	8
M	I	C	H	I	A	M	O	R	I	C	C	A	R	D	O	S	C	O	T	T	I		

Arrivati a questo punto trascriviamo, per due volte di seguito, tutto l'alfabeto internazionale:

ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ

Ora, trascuriamo la frase chiave ed occupiamoci solamente dei numeri.
La prima lettera da codificare corrisponde alla lettera M sopra la quale troviamo il numero 3.
Cerchiamo sui due alfabeti internazionali la lettera M e spostiamoci di 3

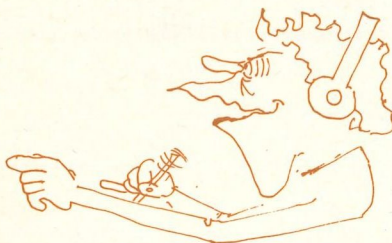
```
100 REM      ********************
120 REM      *                  *
140 REM      *  CODICE CESARE   *
160 REM      *        BY        *
180 REM      * RICCARDO SCOTTI  *
200 REM      *      ISCHIA      *
220 REM      *                  *
240 REM      ********************
260 C=1
280 DIMA$(80),D(2000)
300 PRINT"▒SCRIVI LA FRASE CHIAVE▒":INPUTA$
320 FORI=1TOLEN(A$)
340 A$(I)=MID$(A$,I,1)
360 NEXTI
380 FORI=65TO90
400 B$=CHR$(I)
420 FORJ=1TOLEN(A$)
440 IFB$=A$(J)THENA$(J)=A$(J)+STR$(C):C=C+1
460 NEXTJ,I
480 PRINT:PRINT
500 FORI=1TOLEN(A$)
520 PRINT"▒"A$(I)"■  ";:NEXT
540 PRINT:PRINT"▒▒▒▒VUOI CODIFICARE O DECODIFICARE (C/D) ?"
560 GETX$:IFX$<>"C"ANDX$<>"D"THEN560
580 IFX$="D"THEN1040
600 PRINT"▒▒▒FRASE DA CODIFICARE▒"
620 OPEN3,8,3,"@0:CCES,S,W"
640 Y$="":B$=""
660 GETY$:IFY$=""THEN660
680 PRINTY$;
700 B$=B$+Y$
720 IFLEN(B$)=LEN(A$)THEN780
740 IFY$="0"THEN780
760 GOTO660
780 PRINT:PRINT
800 FORI=1TOLEN(B$)
820 C$=MID$(B$,I,1)
840 IFC$="0"THEN980
860 D(I)=ASC(C$)+VAL(MID$(A$(I),3))
880 IFD(I)>90THEND(I)=D(I)-26
900 D$=D$+CHR$(D(I))
920 PRINTCHR$(D(I));
940 NEXT
960 PRINT:PRINT
980 PRINT#3,D$;CHR$(13);
1000 IFY$="0"THENCLOSE2:CLOSE3:RUN
1020 D$="":GOTO640
1040 PRINT"▒▒▒FRASE DA DECODIFICARE▒"
1060 OPEN2,8,2,"0:CCES,S,R"
1080 INPUT#2,B$
1100 RS=ST:IFRS=64THENCLOSE2:CLOSE3
1120 FORI=1TOLEN(A$)
1140 C$=MID$(B$,I,1)
1160 IFC$=""THENEND
1180 D(I)=ASC(C$)-VAL(MID$(A$(I),3))
1200 IFD(I)<65THEND(I)=D(I)+26
1220 PRINTCHR$(D(I));
1240 NEXT
1260 GOTO1080
```

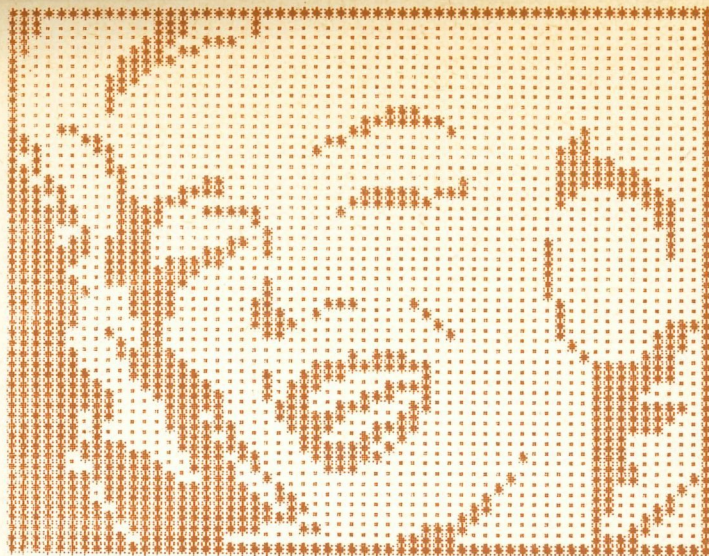
posti verso destra.
La lettera P sara' la prima lettera codificata.
Con questo sistema si procede fino alla fine della frase da "nascondere", e... otterremo:

PZOUAFFIXKAXKGKKPQPXCT

Puo' capitare, a volte, che il testo da codificare possa essere piu' lungo della frase chiave; si proseguira' semplicemente ricominciando dacapo ogni volta che terminano i numeri.

Avrete senza dubbio notato che non vi sono spazi fra le singole parole. Questo fatto deve essere una regola tassativa, come pure quella che vieta l'uso dei numeri e caratteri diversi dall'alfabeto.

Il sistema di decodifica risulta esser esattamente il medesimo, ma invece di contare le lettere sugli alfabeti internazionali procedendo da sinistra verso destra, in questo caso, si procede da destra verso sinistra, ed ecco riapparire la frase in chiaro.

REMark.

280	Dimensionamento della frase chiave che non deve superare in ogni caso i 79 caratteri.
300	La frase chiave puo' essere composta da qualunque parola, un nome o una citazione, purche' non contenga spazi o numeri o caratteri strani.
320-360	Scompone la frase nelle singole lettere e le assegna ad un vettore.
380	65...90 sono i valori ASCII delle lettere dell'alfabeto internazionale.
400-460	Partendo dalla prima lettera dell'alfabeto, cerca nella frase chiave la lettera corrispondente, e le assegna un valore numerico crescente.
620	Apre un canale di scrittura su disco.
660-670	Accetta, una lettera per volta, il testo da codificare e accumula nella variabile B$ fino al raggiungimento della lunghezza della frase chiave. Nel caso si termini anzitempo si dovra' battere il numero 0.
800-940	Questa e' la parte in cui avviene la codifica.
980	Registra su disco la frase codificata.
1060	Apre il canale di lettura da disco.
1100	Alla variabile RS viene assegnato il valore dello "Status". Quando RS assume il valore 64 che corrisponde alla situazione di fine file, il canale di lettura viene, giustamente, chiuso.
1120-1260	In questa parte avviene la decodifica con il procedimento inverso di quello riportato nelle righe 800-940.

M a r i l y n

di Roberto Sozzani.

Chi possiede la stampante CBM 4022 deve necessariamente sostituire le righe 10 e 600 con le seguenti:

```
10 OPEN6,4,6:PRINT#6,CHR$(18):CLOSE6
600 OPEN6,4,6:PRINT#6,CHR$(32):CLOSE6
```

```
10 OPEN6,4,6:PRINT#6,CHR$(14):CLOSE6
20 OPEN4,4:CMD4
30 DIMA$(53)
40 A$(0)="**********************************************************"
50 A$(1)="***.........***.....*....................................*"
60 A$(2)="***.........***.....*....................................*"
70 A$(3)="**.........***..***......................................*"
80 A$(4)="**........***.**.........................................*"
90 A$(5)="*........*****...........................................*"
100 A$(6)="*........***.............................................*"
110 A$(7)="*.......***..............................................*"
120 A$(8)="*.......**...............................................*"
130 A$(9)="*.......*................................................*"
140 A$(10)="*.......*......................***.......................*"
150 A$(11)="*............................*******.....................*"
160 A$(12)="*...**......................**......*..........*.........*"
170 A$(13)="**....**..................**..................**.........*"
180 A$(14)="***....**................*...................***.........*"
190 A$(15)="*.*.....*....................................*****.......*"
200 A$(16)="***.....**...................................*******.....*"
210 A$(17)="**.*.....*......**...................*.......********....*"
220 A$(18)="*****....*....****...........*****.***.......**...****...*"
230 A$(19)="***.*....**.***.............*******..........*......***..*"
240 A$(20)="******...****...*****......*.........................**..*"
250 A$(21)="******...****.......*................................**..*"
260 A$(22)="****..*..***.........*................................*..*"
270 A$(23)="****.**.****......**.*......................*.........*..*"
280 A$(24)="*******.****....***..*......................*.........*..*"
290 A$(25)="******..****.*****..........................*............*"
300 A$(26)="*****...********............................*............*"
310 A$(27)="******..*******......*......................*............*"
320 A$(28)="*******...****.......*......................*............*"
330 A$(29)="******....****......**....***....*...........*...........*"
340 A$(30)="******....****......***..*........*..........*...........*"
350 A$(31)="******.....**.......****...........*.........*........****"
360 A$(32)="*****...*..**........**.............*........*.......***.*"
370 A$(33)="*****....*.**.................................*......***.*"
380 A$(34)="******...*****..............*.***..............*....***..*"
390 A$(35)="******....*****..........**********.............*******..*"
400 A$(36)="*******...******.....*..****......*.............*******..*"
410 A$(37)="********...******....*.***.....****.............*****....*"
420 A$(38)="*********...******...*.**....***..*.............****.....*"
430 A$(39)="*********....*****....***..***...**.............********.*"
440 A$(40)="******.**....***.*....******....**..............*******..*"
450 A$(41)="******.*.....******....****...****..............******...*"
460 A$(42)="******.*......***.*.....**...**.*...............***......*"
470 A$(43)="**********....******....******.*................***......*"
480 A$(44)="*****.******...***.**....******...........*.....***......*"
490 A$(45)="*****..******..****.*.....***...................****.....*"
500 A$(46)="**************..***..*...................*......****....**"
510 A$(47)="*******.************..*.................*.......**.....*.*"
520 A$(48)="*******.*************.**...............*........**....*..*"
530 A$(49)="********.*************.**.............**........**...**..*"
540 A$(50)="************************.............**.........*...**...*"
550 A$(51)="*********.***********...**........****..........*..**....*"
560 A$(52)="**********.*******..............*****...........****.....*"
570 A$(53)="**********************************************************"
580 FORI=0TO53:PRINTTAB(10)A$(I):NEXT
590 PRINT#4:CLOSE4
600 OPEN6,4,6:PRINT#6,CHR$(24):CLOSE6
```

Flussi RELative

di
Gloriano Rossi

Sul numero scorso di POCKET PET abbiamo visto come il PET-CBM organizza i file su disco, in particolare abbiamo trattato dei flussi ad organizzazione RELative.

In questa seconda puntata vedremo come creare un flusso relative, come aggiornarlo e come modificarne i contenuti.

Su un numero recente di una rivista inglese e' apparso un interessante programma studiato appositamente per poter eseguire una sofisticata gestione dei files relative.
Il listato riportato sulla rivista era pero' zeppo di errori e quindi il programma non girava assolutamente.

Visto pero' che l'idea poteva essere estremamente valida, dopo una accurata analisi e molte correzioni ecco l'edizione italiana di FIXIT.

Prima di passare ad una descrizione di massima del programma FIXIT, e' bene spiegare nella giusta maniera cosa si intende dire con il termine "Programma di utilita'".
Un programma di utilita' e' composto da una serie di istruzioni che prevedono funzioni ed effetti che si adattano facilmente a molteplici usi e casi e che il loro uso viene sfruttato quale coadiuvante di attivita' gestionali o di specifiche applicazioni.

Il FIXIT svolge eggregiamente la funzione di editing di un qualsiasi file organizzato in maniera relative e proprio per questa sua malneabilita' che puo' essere tranquillamente annoverato fra i programmi di utilita' di alta qualita'.

Il programma FIXIT.

Un qualsiasi file organizzato in maniera relative puo' essere consultato da FIXIT.

FIXIT infatti e' parametrizzato in maniera tale che e' possibile fornire di volta in volta sia il nome del file in oggetto che deve essere consultato che la caratteristica di lunghezza record in esso contenuto.

Quando saranno forniti questi due parametri basilari, FIXIT esibisce una mascherina che comprende tante caselline quanti sono i bytes che compongono ogni singolo record.

Per ragioni di dimensione di schermo, questa mascherina e' disegnata automaticamente fino ad un massimo di 90 caselline corrispondenti a 90 caratteri del record.

I files relative possono avere, pero', records con lunghezza superiore ai 90 caratteri citati. Proprio per questa ragione FIXIT provvede ad evidenziare una sola parte di ogni record preso in esame.
La parte di record visualizzata dovra' essere definita dall'utente al momento successivo dell'imputazione dei primi due dati obbligatori.
La possibilita' di definire la parte di record da visualizzare avviene solamente quando la lunghezza del record supera i 90 caratteri, in caso contrario FIXIT bypassera' questa facility in taluni casi necessaria.
FIXIT, in questi casi, chiedera' da quale byte e per quanti bytes dovra' visualizzare il record; la visualizzazione parziale in ogni caso non potra' essere superiore ad 80 caratteri per volta.

Un file organizzato in maniera relative preso in esame da FIXIT puo' essere consultato in due modi:

1) Accesso sequenziale del file.
Dopo aver impostato il numero relativo del primo record da visualizzare e' sufficente premere il tasto corrispondente alla freccia verso l'alto perche' sia visibile il record immediatamente sucessivo.
2) Accesso ad uno qualsiasi singolo record.
Ogni qualvolta sia necessario visualizzare uno specifico record sara' necessario semplicemente premere il tasto corrispondente alla A commerciale ("chiocciolina" in gergo); FIXIT domandera' allora quale sia il numero relativo del record richiesto.

Ogni record sara' visualizzato con un carattere per ogni casella della maschera ed un piccolo segno grafico indica la casellina ove e' simulato il cursore; a questo punto un record, cosi' presentato, puo' essere corretto o creato "ex novo".
L'editing e' reso possibile digitando i caratteri necessari al componimento del record oppure muovendo il cursore con i normali tasti di muovimento di schermo.
Come avrete senzaltro intuito questi tasti sono: cursore a destra, cursore a sinistra, cursore su', cursore in giu', delete carattere, insert carattere.

Una delle particolarita' interessanti di FIXIT e' quella di consentire la visualizzazione, la creazione e/o l'aggiornamento di ogni singolo record di un file relative. Il record viene preso in esame nella sua completezza, viene cioe' visualizzato per intero, campo per campo.
Si sa infatti, cosi' ci insegna il manuale Commodore dei Floppy Disk, che ogni record di un file relative puo' essere composto da vari campi divisi ciascuno dal carattere ASCII 13. In fase di lettura normale, cioe' con lo statment BASIC di input, il sistema puo' considerare solamente quella parte di record che va dall'inizio (definito o no) fino al primo carattere ASCII 13.
La routine di input di FIXIT, invece, prevede l'ingresso del record utilizzando la funzione GET, cio' ci permette di poter prendere in esame qualsiasi carattere ASCII del record ed intervenire in conseguenza.

F I X I T

```
100 REM ********************************
110 REM *                              *
120 REM *          F I X I T           *
130 REM *                              *
140 REM *  EDITOR PER FILES RELATIVE   *
150 REM *                              *
160 REM ********************************
170 REM *                              *
180 REM *        POCKET GROUP          *
190 REM *       GLORIANO ROSSI         *
200 REM *                              *
210 REM ********************************
220 REM
230 Z=1/254
240 G$(0)="   ┌──┬──┬──┬──┬──┬──┬──┬──┬──┬──┐
250 G$(1)="   │  │ 0│1 │2 │3 │4 │5 │6 │7 │8 │9 │
260 G$(2)="   ├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
270 G$(3)="   │  │  │  │  │  │  │  │  │  │  │  │
280 GM$="   ├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
290 GB$="   └──┴──┴──┴──┴──┴──┴──┴──┴──┴──┘
300 AC$="": D$="": SP$="      ": AD$=""
310 FORK=1TO5: SP$=SP$+SP$: AD$=AD$+D$: AC$=AC$+AC$: NEXT
320 C7$=LEFT$(AC$,7): SP$=SP$+LEFT$(SP$,100)
330 PP$="______________": TM$=CHR$(13)
340 GOSUB2510
350 INPUT"INPUT FILENAME   #   ";A$
360 IFA$="F"THENPRINT"";:END
370 IFA$<>"*"THENNF$=A$
380 INPUT"LUNGHEZZA DEL RECORD ";RL
390 DOPEN#1,(NF$),L(RL),D0
400 IFRL>90THEN840
410 FB=1:LB=RL:O$="-"
420 GOSUB1770
430 GOSUB700:INPUT"RECORD #   *   ",R$
440 GOSUB760:IFR$="*"THENDCLOSE:GOTO350
450 REC=VAL(R$)
460 RECORD#1,(REC):GOSUB700
470 GOSUB2070
480 PRINT"RECORD # "REC;
490 GOSUB1900:POKE158,0
500 GETA$:IFA$=""THEN500
510 IFA$=""THEN970
520 IFA$=CHR$(20)THEN1040
530 IFA$=""THEN1110
540 IFA$=""THEN1190
550 IFA$=""THEN1270
560 IFA$=""THEN1350
570 IFA$="@"THENPRINTO$:GOTO420
580 IFA$="↑"THENREC=REC+1:PRINTO$:GOTO460
590 IFA$=CHR$(34)THENA$="@":GOTO1430
600 IFA$=CHR$(141)THEN1700
610 IFA$=CHR$(13)THENA$="":GOTO1430
620 IFA$=""THEN400
630 IFA$=""THENDCLOSE:PRINT"";:END
640 GOTO1430
650 REM
660 REM ********************************
670 REM * CURSORE ALLA LINEA N. 24     *
```

```
680 REM ******************************
690 REM
700 PRINT"▒"AD$LEFT$(SP$,30)"▒"AD$;:RETURN
710 REM
720 REM ******************************
730 REM *  TEST DS (DISK ERROR)       *
740 REM ******************************
750 REM
760 IFDS<20THENRETURN
770 IFDS=50THENRETURN
780 PRINT"▒"DS$:DCLOSE:END
790 REM
800 REM ******************************
810 REM * SET RANGE PER INSPEZIONE   *
820 REM ******************************
830 REM
840 INPUT"▒PRIMO BYTE DA VISUALIZZARE ";A$
850 IFA$=" "THENDCLOSE:GOTO350
860 FB=VAL(A$)
870 IFRL-FB<81THENLB=RL:GOTO420
880 INPUT"NR DI BYTES DA VISUALIZZARE ";A$
890 A=VAL(A$):IFA>80THENPRINT"RANGE TROPPO GROSSO!":GOTO880
900 IFFB+A>RLTHENLB=RL:GOTO420
910 LB=FB+A-1:GOTO420
920 REM
930 REM ******************************
940 REM *        INSERIMENTO         *
950 REM ******************************
960 REM
970 B2$=LEFT$(" "+B2$,LEN(B2$))
980 GOSUB1900:GOTO500
990 REM
1000 REM ******************************
1010 REM *         DELETE             *
1020 REM ******************************
1030 REM
1040 B2$=MID$(B2$,2)+" "
1050 GOSUB1900:GOTO500
1060 REM
1070 REM ******************************
1080 REM *      CRSR  SINISTRA        *
1090 REM ******************************
1100 REM
1110 IFLEN(B1$)=0ORBN=FBTHEN500
1120 B=1:GOSUB1500
1130 GOTO500
1140 REM
1150 REM ******************************
1160 REM *      CRSR  DESTRA          *
1170 REM ******************************
1180 REM
1190 IFLEN(B2$)=0ORBN>LBTHEN500
1200 B=1:GOSUB1590
1210 GOTO500
1220 REM
1230 REM ******************************
1240 REM *      CRSR  SU'             *
1250 REM ******************************
1260 REM
1270 IFLEN(B1$)<10ORBN<FB+10THEN500
```

```
1280 B=10:GOSUB1500
1290 GOTO500
1300 REM
1310 REM ******************************
1320 REM *       CRSR  GIU'           *
1330 REM ******************************
1340 REM
1350 IFLEN(B2$)<10ORBN>LB-10THEN500
1360 B=10:GOSUB1590
1370 GOTO500
1380 REM
1390 REM ******************************
1400 REM * PRINT A$ & MOVE MARKER     *
1410 REM ******************************
1420 REM
1430 PRINT"▮─▮▮"A$"▮▮";:B2$=A$+MID$(B2$,2)
1440 GOTO1190
1450 REM
1460 REM ******************************
1470 REM * JSR X CRSR SINISTRA        *
1480 REM ******************************
1490 REM
1500 PRINTO$:BN=BN-B:GOSUB2250
1510 B2$=RIGHT$(B1$,B)+B2$
1520 B1$=LEFT$(B1$,LEN(B1$)-B)
1530 RETURN
1540 REM
1550 REM ******************************
1560 REM * JSR X CRSR DESTRA          *
1570 REM ******************************
1580 REM
1590 PRINTO$
1600 BN$=RIGHT$(STR$(BN),1):IFBN$<>"9"THENPRINT"▮▮▮"
1610 BN=BN+B:GOSUB2250
1620 B1$=B1$+LEFT$(B2$,B)
1630 B2$=MID$(B2$,1+B)
1640 RETURN
1650 REM
1660 REM ******************************
1670 REM * RIMPIAZZA RECORD (ENTER)   *
1680 REM ******************************
1690 REM
1700 PRINTO$:GOSUB2360
1710 REC=REC+1:GOTO460
1720 REM
1730 REM ******************************
1740 REM * PRINT GRIGLIA SU SCHERMO   *
1750 REM ******************************
1760 REM
1770 PRINT"▮"
1780 FORK=0TO3
1790 :PRINTG$(K):NEXT
1800 FORK=INT(FB/10)TOLB/10
1810 N$=RIGHT$(STR$(K),2)
1820 GN$="   ▮|"+N$+"|▮   |  |  |  |  |  |  |  |  |  |
1830 PRINTGN$:PRINTGM$:NEXT
1840 PRINT"▮"GB$:RETURN
1850 REM
1860 REM ******************************
1870 REM * METTE B2$ SU SCHERMO       *
```

```
1880 REM ******************************
1890 REM
1900 BN=LEN(B1$)+1:GOSUB2250
1910 PRINT"▚┣▜▌";:R1=0
1920 FORK=1TOLB-BN+1
1930 :BY$=MID$(B2$,K,1)
1940 :PRINTBY$"▚▌";:PS=PS+1:R1=R1+1
1950 :IFR1=9THENPS=0
1960 :IFPS=10THENPS=0
1970 :IFPS=0THENPRINT:PRINT:PRINTC7$;
1980 NEXT
1990 GOSUB2250
2000 RETURN
2010 REM
2020 REM ******************************
2030 REM * LEGGE RECORD INTO A1$      *
2040 REM * LO DIVIDE IN B1$ E B2$     *
2050 REM ******************************
2060 REM
2070 GOSUB760:A1$=""
2080 FORK=1TORL
2090 :GET#1,B$
2100 :IFB$=CHR$(13)THENB$="◥"
2110 :IFB$=CHR$(34)THENB$="@"
2120 :A1$=A1$+B$:IFST=64THENK=RL
2130 NEXT
2140 B1$="":A=LEN(A1$)
2150 IFA<RL-1THENA1$=A1$+LEFT$(SP$,RL-A)
2160 IFFB>1THENB1$=LEFT$(A1$,FB-1)
2170 B2$=(MID$(A1$,FB))
2180 RETURN
2190 REM
2200 REM ******************************
2210 REM * POSIZIONE CURSORE IN BN E *
2220 REM * PRINT "┴" SULLA GRIGLIA   *
2230 REM ******************************
2240 REM
2250 A=INT(BN/10+Z):R=A-INT(FB/10+Z):REM RIGA
2260 PS=INT((BN/10-A)*10+Z)          :REM COLONNA
2270 R=6+2*R:C=7+2*PS
2280 PRINT"▗"LEFT$(AD$,R)LEFT$(AC$,C);
2290 PRINT"┴▌▛";
2300 RETURN
2310 REM
2320 REM ******************************
2330 REM * LEGGI B1$ E B2$ NEL RECORD*
2340 REM ******************************
2350 REM
2360 RECORD#1,(REC):GOSUB760:B$=""
2370 A1$=B1$+B2$
2380 FORK=1TOLB
2390 :A$=LEFT$(A1$,1):A1$=MID$(A1$,2)
2400 :IFA$="◥"THENA$=CHR$(13)
2410 :IFA$="@"THENA$=CHR$(34)
2420 :B$=B$+A$
2430 NEXT
2440 PRINT#1,B$;:GOSUB760
2450 RETURN
2460 REM
2470 REM ******************************
```

```
2480 REM *        ISTRUZIONI          *
2490 REM ****************************
2500 REM
2510 POKE59468,14:PRINT"""TAB(12)LEFT$(PP$,9)TM$TAB(12)"** FIXIT *"
2520 PRINTTM$"QUESTO E' UN EDITOR PER FILES RELATIVE."
2530 PRINT"FILES CON PIU' DI 90 BYTES SARANNO VI-"
2540 PRINT"SUALIZZATI PER SEZIONI DI < 80 BYTES"TM$
2550 PRINT"IL BYTE IN EDIT E' MARKATO DA '┴'
2560 PRINT"QUESTO MARKER E' DA MUOVERE NELLA
2570 PRINT"MASCHERA"TM$
2580 PRINT"I COMANDI OPERATIVI SONO:"TM$
2590 PRINT" @    - CAMBIO RECORD#
2600 PRINT" π    - PROSSIMO RECORD
2610 PRINT" HOME - CAMBIO DEL BYTE DI RANGE
2620 PRINT" CLR  - FINE ESECUZIONE"TM$
2630 PRINT" SHIFT RETURN  - ENTRATA MODO EDIT
2640 PRINT" #    - CAMBIO FILE"TM$
2650 PRINT"IL RETURN E' EVIDENZIATO DAL CARATTER ◥
2660 PRINT"E I DOPPI APICI DAL CARATTERE @
2670 PRINT"■■■■■■BATTI UN TASTO PER CONTINUARE!"";
2680 POKE158,0:WAIT158,1
2690 POKE158,0:PRINT"":RETURN
```

Per merito ed in conseguenza di questo fatto sullo schermo ayremo due casi di trasformazione di carattere.
Quando FIXIT incontra un carattere ASCII 13 questo viene tradotto e visualizzato con un carattere grafico univoco; il medesimo criterio di trasformazione viene eseguito anche per il carattere corrispondente ai doppi apici; questo carattere viene visualizzato con la "chiocciolina" (A commerciale).

FIXIT oltre che consultare e modificare il contenuto puo' anche creare un file ad organizzazione relative.
Un file relative creato "ex novo" permette di definire sia il nome del file stesso che il dimensionamento del record.
E' chiaro che la creazione di un record con piu' di 90 caratteri comporta due o piu' mascherate.
La dimensione del file e' definita automaticamente nel momento in cui si inserisce l'ultimo record (introduzione sequenziale) oppure quando ci si posiziona, come prima operazione, sul numero di record che si reputa di raggiungere.

Vediamo di fare un esempio pratico per spiegare questi due concetti:
Dobbiamo creare un file relative di 100 record con 50 caratteri ciascuno. La prima domanda di FIXIT e' relativa al nome del file; se battiamo ad esempio "PROVA" e poi alla domanda inerente alla lunghezza record si battera' 50, FIXIT andra' ad esaminare il drive 0 e controllera' l'esistenza del file. Se FIXIT non trova corrispondenza nella directory il led rosso di errore si accende, ma il programma prosegue regolarmente; questa e' una condizione prevista per la creazione di un file.
Nel caso in cui il file "PROVA" esistesse gia' precedentemente FIXIT andra' ad esaminare, al primo acceso ad un record, le caratteristiche di lunghezza del record stesso e se trovasse una discordanza cio' provocherebbe una condizione di errore effettivo (70,NO CHANNEL,00,00); avremo tentato di eseguire un edit su un file mal dichiarato.

Torniamo ora alla creazione di file ed alla domanda inerente al numero relativo di record scriveremo il numero 100 e quindi batteremo il tasto RETURN contemporaneamente al tasto SHIFT.
Cosi' facendo l'unita' disco girera' e quindi verra' visualizzato l'ipotetico record 101; a questo punto dopo aver premuto il tasto "chiocciolina" potremo in seguito posizionarci su qualsiasi record desiderato. Se invece premiamo il tasto CLR usciremo dal FIXIT ed avremo creato un file relative di nome "PROVA" completamente vuoto composto da 100 records da 50 caratteri ciascuno.

Similarmente, si puo' procedere per l'ampliamento di un file. Si procede puntando sul record nnn che corrispondera' all'ultimo nuovo record.
Questa nuova facility non distrugge i precedenti records gia' introdotti in precedenza, ma allarga il numero di record disponibili nel file.

Arrivati a questo punto penso che meglio delle mie parole possa la pratica. Quindi digitate il FIXIT ed eseguite le opportune prove.

Per finire, quale appuntamento ed anticipazione, diro' che la prossima puntata trattero' il tema della tecnica di randomizzazione.

SPIGOLATURE

```
10 REM ********************************
20 REM ***      PALLINA  ROMBO     ***
30 REM ********************************
40
100 PRINT"▐▮▮"
110 A=INT(RND(1)*15):B=INT(RND(1)*15):PRINT"▮▮  ";
120 FORK=1TOA:PRINT"▮▮ ▮▮";:NEXT
130 FORK=1TOB:PRINT"▮▮ ▮▮▮▮";:NEXT
140 FORK=1TOA:PRINT"▮▮ ▐▮▮▮";:NEXT
150 FORK=1TOB:PRINT"▮▮ ▐▮";:NEXT
160 GOTO110
```

```
10 REM********************************
20 REM***    PALLINA A CASO     ***
30 REM********************************
40
100 PRINT"▐▮▮"
110 X=INT(RND(1)*8): IFX=YTHEN110
120 A=INT(RND(1)*15):B=INT(RND(1)*15):PRINT"▮▮  ";
130 ON X GOSUB 140,160,150,160,150,170,170:Y=X:GOTO110
140 FORK=1TOA:PRINT"▮▮ ▮▮";:GOSUB 190:NEXT:RETURN
150 FORK=1TOB:PRINT"▮▮ ▮▮▮▮";:GOSUB 190:NEXT:RETURN
160 FORK=1TOA:PRINT"▮▮ ▐▮▮▮";:GOSUB 190:NEXT:RETURN
170 FORK=1TOB:PRINT"▮▮ ▐▮";:GOSUB 190:NEXT:RETURN
180 GOTO120
190 N=10:FORH1TON:NEXT:RETURN
```